EngSci Year 3 Fall 2022 Notes

Brian Chen

Division of Engineering Science
University of Toronto
https://chenbrian.ca

brianchen.chen@mail.utoronto.ca

Contents

I ECE349: Introduction to Energy Systems 1

1 Admin stuff 1
1.1 Lecture 1 1
1.1.1 Mark breakdown 1

2 AC Steady State Analysis 1
2.1 Lecture 2 1
2.1.1 TODO 1
2.2 Lecture 3 3

3 AC Power 4
3.1 Lecture 4 5
3.1.1 Root Mean Squared (RMS) Values 6
3.2 Lecture 5: Multi-Phase AC 7
3.3 Lecture 6: Y and Delta connections 10
3.4 Lecture 7: DC-DC conversion 12
3.5 Lecture 8: Average output voltage 15
3.5.1 Filtering 17
3.6 Lecture 9 18
3.6.1 Inductor VoH-Seconds balance 20
3.6.2 Capacitor charge balance 20
3.7 Lecture 10 21
3.7.1 Design Process 23
3.8 Lecture 11 24
3.9 Lecture 12 26

II ECE352: Computer Organization 26

4 Admin stuff 26
4.1 Lecture 1 26
4.1.1 Mark breakdown 26

5 Preliminary 26
5.1 Lecture 2: Using binary quantities to represent other things 26
5.1.1 Floating Point Numbers 27

6 NIOS II Preliminary **28**
6.1 Lecture 3: Behavioural Model of Memory 28
6.1.1 Memory 29
6.1.2 Physical Interface 30
6.2 Lecture 4: NIOS II Programming Model 31
6.2.1 Adding Two Numbers 32
6.2.2 Adding two numbers using memory 34

7 Assembly Basics **36**
7.1 Lecture 5: Simple Control Flow 36
7.2 Lecture 6, 7: For loops and arrays 38
7.3 Lecture 8: Subroutines 40
7.4 Lecture 9 46
7.5 Lecture 10: Recursive Subroutines 48
7.6 Lecture 11: Structs and recursive structures 49
7.7 Lecture 12: Devices 51
7.8 Lecture 12 52

III ECE355: Signal Analysis and Communication **52**

8 Admin and Preliminary **53**
8.1 Lecture 1 53
8.1.1 Mark Breakdown 53

9 Transformations **53**
9.1 Lecture 2 53
9.2 Lecture 3 54
9.2.1 General Continuous Complex Exponential Signals 55

10 Basic Signals **55**
10.1 Lecture 4: Step and Impulse Functions 55
10.2 Lecture 5 57
10.2.1 Types of systems 58
10.2.2 System properties 59

11 LTI systems **60**
11.1 Lecture 6: Discrete LTI systems 60
11.2 Lecture 7: Continuous LTI systems 62
11.3 Lecture 8 62

IV ECE360: Electronics **62**

12 Admin and Preliminary 63
12.1 Lecture 1 63
12.1.1 Mark Breakdown 63
12.1.2 Diodes 63

13 Diodes 64
13.1 Lecture 2 64
13.2 Lecture 3 66

14 Lecture 4 & 5: Forward conducting diodes 68

15 Small-Signal Model 70

16 Lecture 6: Small signal model, cont'd 71
16.0.1 Deriving small-signal resistance 72

17 Lecture 7 74

18 Rectifiers 75
18.1 Lecture 8 77
18.2 Lecture 9 80

V ECE358: Foundations of Computing 83

19 Admin and Preliminary 83
19.1 Lecture 1 83
19.1.1 Mark Breakdown 83

20 Complexities 83
20.1 Lecture 2 83
20.2 Lecture 3: Logs & Sums 86
20.2.1 Functional Iteration 86
20.3 Lecture 4: Induction & Contradiction 87
20.3.1 Induction 87
20.3.2 Contradiction 88
20.4 Lecture 5: recurrences 89
20.4.1 Recurrence Trees 90
20.4.2 Substitution 91
20.4.3 Master Theorem 91
20.5 Lecture 6 92
20.5.1 Graphs 92
20.5.2 Trees 94
20.6 Lecture 7: Probability and Counting 94

VI MAT389: Complex Analysis 95

21 Complex Numbers **95**
21.1 Lecture 1 95
21.2 Lecture 2 97
21.2.1 Functions on complex planes 98
21.2.2 Exponential Functions 99
21.3 Lecture 3: Exponent and Logarithm 101
21.3.1 Exponential 101
21.3.2 Logarithm 102
21.3.3 Powers 103
21.4 Lecture 4 103
21.4.1 Properties of complex derivative 105
21.5 Power series 109
21.6 Lecture 5 110

VII ECE444: Software Engineering **110**

22 Preliminary **110**
22.1 Lecture 1, 2 111

23 Project Management **111**
23.1 Lecture 3 111
23.1.1 Agile 112
23.2 I dropped this course 112

VIII ESC301: Seminar **112**

24 Preliminary **113**
24.1 Seminar 1 113

ECE349: Introduction to Energy Systems

PART

I

Taught by Prof. P. Lehn

Section 1

Admin stuff

Subsection 1.1

Lecture 1

First lecture was logistical info + a speil about how power systems are one of the great modern wonders. Course will cover sinusoidal AC power systems (1, 3 phase), power systems (dc-dc, dc-ac conversion), and magnetic systems (transformers, actuators, and synchronous machines)

1.1.1 Mark breakdown

- 50 % Final
- 25 % Midterm
- 5 % Quiz
- 15 % Labs
- 5 % Assignments

Section 2

AC Steady State Analysis

Subsection 2.1

Lecture 2

2.1.1 TODO

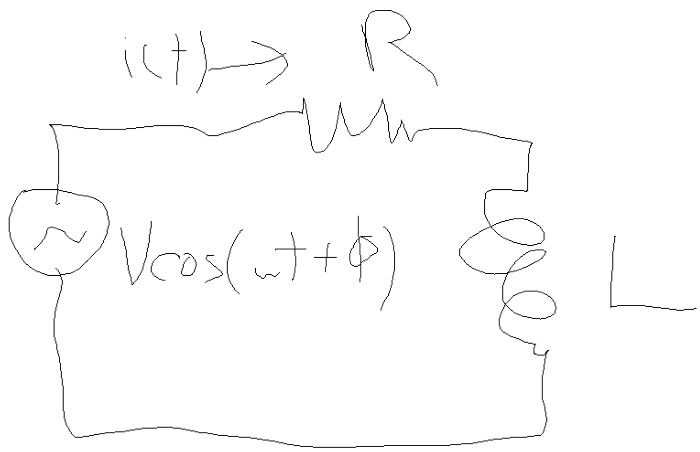
- Review Thomas 669-600

Recall, for a general phasor $\hat{P}$

- $\frac{d\hat{P}}{dt} = jw\hat{P}$
- $\int \hat{P} = \frac{1}{jw}\hat{P}$

What we have learnt prior for differential equations enables us to arrive at analytical solutions to linear stable AC systems with phasors. A homogeneous and particular solution will be produced. If there's a stable homogeneous solution, $\rightarrow 0$ as $t \rightarrow \infty$. The full solution would be the addition of the two via super position.

We generally use this approach to solve circuits since it's an efficient way to solve circuits and make them into essentially DC circuits.



$$Ri + L\frac{di}{dt} = V\cos(\omega t + \phi) \tag{2.1}$$

But this is a pain to solve. It can be made simpler by applying phasors

$$V\cos(\omega t + \phi) = Re\{Ve^{j(\omega t + \phi)}\} \tag{2.2}$$

Take the real part of $\hat{I}$:

$$R\hat{I} + L\frac{d\hat{I}}{dt} = Ve^{j(\omega t + \phi)} \tag{2.3}$$

And therefore by inspection the solution is of format $\hat{I}e^{j\omega t}$, where $\hat{I}$ is a phasor. Noting that $\hat{I}$ contains only amplitude and phase,

$$\begin{gathered} R\hat{I}e^{j\omega t} + L\frac{d}{dt}(\hat{I}e^{j\omega t}) = Ve^{j\omega + \phi} \\ R\hat{I} + L\hat{I}j\omega = Ve^{j\phi} \end{gathered} \tag{2.4}$$

And now reconstructing:

$$\begin{aligned} \hat{I} &= \frac{V}{\sqrt{R^2(\omega L)^2}} e^{j(\omega t + \phi - \tan^{-1}\frac{wL}{R})} \\ i(t) &= Re\left\{\hat{I}\right\} \end{aligned} \tag{2.5}$$

And therefore

$$\hat{I} = \frac{V}{\sqrt{R^2(\omega L)^2}} \cos(\omega t + \phi - \tan^{-1}(\frac{wL}{R})) \tag{2.6}$$

The steps to solving a phasor problem are:

Notation: $Xe^{j\phi} \leftrightarrow X\lfloor\phi$

- Define phasor: $V\cos(\omega t + \phi) \leftrightarrow V\lfloor\phi$
- Map L, C into phasor domain; find impedences
  - $v = L\frac{di}{dt} \leftrightarrow \hat{V} = j\omega L\hat{I}$

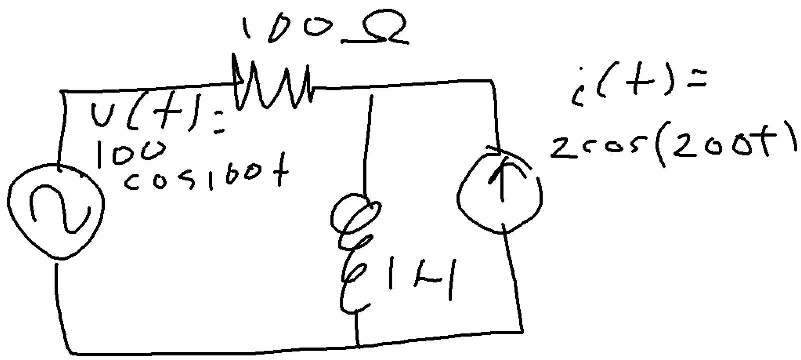
  - $i = C\frac{dv}{dt} \leftrightarrow \hat{V} = \frac{1}{j\omega C}\hat{I}$

- Do mesh analysis to find $\hat{I}$; $\hat{I} = \frac{\hat{V}}{\sum \text{impedances}}$
- Reconstruct $i(t)$ from $\hat{I}$

Subsection 2.2

Lecture 3

Phasors allow us to solve circuits with multiple sources of differing frequencies.



To find the current $i(t)$ over the inductor we can find it's response due to the voltage and current sources and then apply superposition.

- $I_1 = \frac{100\lfloor 0}{1oo+j100} = 0.707\lfloor -45^o \rightarrow i_1(t) = 0.707cos(100t - 45^o)$
- $I_2 = \frac{100}{100+j200}2\lfloor 0 = 0.894\lfloor -65^o \rightarrow i_2(t) = 0.894cos(200t - 63^o)$
- $i(t) = i_1(t) + i_2(t) = 0.707cos(100t - 45^o) + 0.894cos(200t - 63^o)$

Non-sinusoidal stimulus may be solved by decomposing the signal with Fourier transforms. For example, square waves:

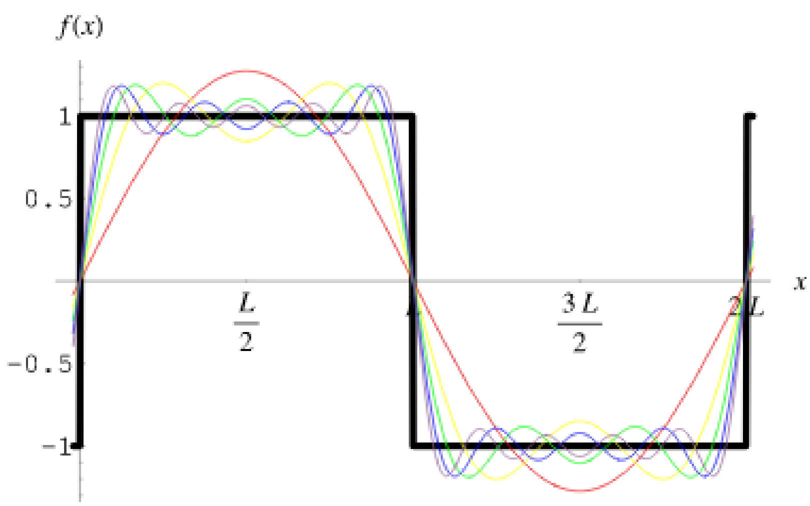


Figure 1. square waves with Fourier series superimposed

The general form of a Fourier transform is given as:

$$v_{equiv}(t) = a_o + \sum_{n=1}^{\infty} a_k cos(nw_o t) + b_k cos(nw_o t) \tag{2.7}$$

Where:

$$\begin{aligned} a_o &= \frac{1}{T}\int_0^T v(t)dt \\ a_k &= \frac{2}{T}\int_0^T v(t)cos(nw_o t)dt \\ b_k &= \frac{2}{T}\int_0^T v(t)sin(nw_o t)dt \end{aligned} \tag{2.8}$$

Armed with Fourier series and superposition we may now model a non-sinusoidal signal as a superposition of an infinite sum of sources. About half the work can be cut in half by recognizing that sin lags cos by 90^o, so

$$\begin{aligned} a_o &= \frac{1}{T}\int_0^T v(t)dt \\ a_k &= \frac{2}{T}\int_0^T v(t)cos(nw_o t)dt \\ b_k &= \frac{2}{T}\int_0^T v(t)cos(nw_o t - 90^o)dt \end{aligned} \tag{2.9}$$

Section 3

AC Power

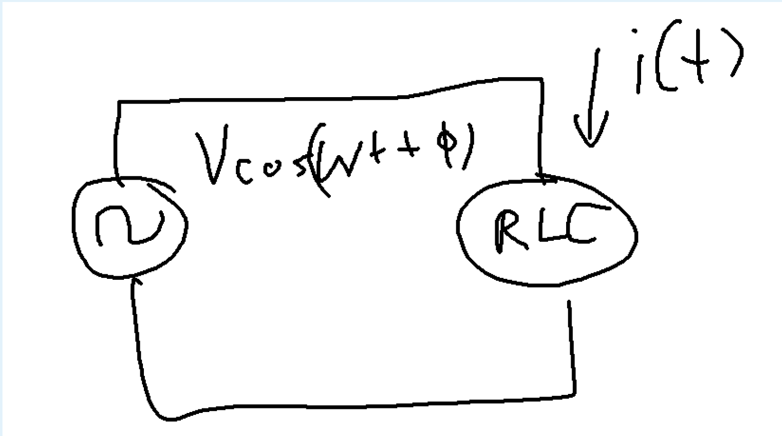
Definition 1 **Instantaneous Power**: $p(t) = v(t) \times i(t)[W, \frac{J}{s}]$

Example For a circuit with a voltage source, $v(t) = Vcos(\omega t)$ and a resistor Ω, $i(t) = Icos(\omega t)$, $p(t) = VIcos^2(\omega t) = \frac{VI}{2}(1 + cos(2\omega t))$

Definition 2 **Average Power over Cycle**: $P(t) = \frac{1}{T}\int_o^T p(t)dt = \frac{VI}{2}$

If we were to plot the instantaneous power we see that due to the sinusoidal response there are times where 0 power is supplied. This will always be true for a single phase power supply; real-world supplies always have multiple phases; this is why computer PSUs always contain a ton of capacitors.

Definition 3 **Reactive Power**: Q



If $\phi_i = 0$ and taking $\phi = \phi_v$,

$\phi = \phi_v - \phi_i$

$$\begin{aligned} p(t) &= V\cos(\omega t + \phi) * I\cos(\omega t) \\ &= \frac{VI}{2}\cos(\phi) + \cos(2\omega t + \phi) \\ &= \underbrace{\frac{VI}{2}\cos(\phi)(1 + \cos(2\omega t)}_{\text{real power e.g. heat}} - \underbrace{\frac{VI}{2}(\sin\phi\sin 2\omega t)}_{\text{stored and released back to source}} \end{aligned} \tag{3.1}$$

Taking the average power of the reactive power we get

$$P_{avg} = \frac{VI}{2}\cos\phi \tag{3.2}$$

Another quantity, reactive power, can be defined with regards to the energy sloshing back and forth:

$$Q = \frac{VI}{2}\sin\phi \tag{3.3}$$

Subsection 3.1

Lecture 4

Definition 4

Displacement factor

$DF \equiv \cos\phi$

Where ϕ is the angle measured from the $\hat{V}$ to the $\hat{I}$ phasors. A $DF = 1$ means the system is transferring the most power possible.

- Lagging DF: ϕ is -'ve
- Leading DF: ϕ is +'ve'

We can also write P, Q from phasors.

$$
\begin{aligned}
P &= \frac{\hat{V}\hat{I}}{2}\cos(\phi_v - \phi_i) \\
&= \frac{1}{2}\hat{V}\hat{I}Re\{e^{j\phi_v - \phi_i}\} \\
&= \frac{1}{2}Re\{Ve^{j\phi v} \cdot Ie^{-j\phi_i}\} \\
&= \frac{1}{2}Re\{\hat{V}\hat{I}^*\}
\end{aligned}
\tag{3.4}
$$

$$
\begin{aligned}
Q &= \frac{\hat{V}\hat{I}}{2}\sin(\phi_v - \phi_i) \\
&\vdots \\
&= \frac{1}{2}Im\{\hat{V}\hat{I}^*\}
\end{aligned}
\tag{3.5}
$$

The expressions for Q and P are basically the same so we define a new quantity, complex power, which incorporates both the imaginary and real components.

Reference: Thomas 16.1, 16.2, 16.3

Definition 5 **Complex Power**

$$S = \frac{1}{2}\hat{V}\hat{I}* = P + jQ \tag{3.6}$$

3.1.1 Root Mean Squared (RMS) Values

A RMS value measures the average power of a periodic signal. Consider a circuit with an AC voltage source with $v(t)$ flowing through a resistor.

The power is:

$$p(t) = v(t)i(t) = v(t)\frac{v(t)}{R} = \frac{1}{R}v(t)^2 \tag{3.7}$$

The average power is therefore

$$P = \frac{1}{R}\int_0^T v(t)^2 dt \tag{3.8}$$

Evaluating this becomes easier by defining an useful quantity, RMS voltage

$$v_{rms} = \sqrt{\frac{1}{T}\int_0^T v(t)^2 dt} \tag{3.9}$$

And using v_{rms} we get a nice expression for average power,

$$P = \frac{1}{R}v_{rms}^2 \tag{3.10}$$

More generally speaking we can define RMS values of sinusoidal signals

Definition 6

$$v_{rms} = \sqrt{\frac{1}{2\pi}\int_0^{2\pi}(\hat{V}\cos wt)^2 dt} = \frac{1}{\sqrt{2}}\hat{V} \tag{3.11}$$

Plugging this into the expressions for complex power:

$$S = \underline{V} \cdot \underline{I^*} \tag{3.12}$$

Where $\underline{V}$, $\underline{I^*}$ are RMS phasors at a given common frequency.

And more generally yet, the RMS values of non-sinusoidal signals can be found with help of a Fourier expansion

Definition 7 Let $v(t) = \hat{v}_0 + \hat{v}_1 \cos(wt + \phi_1) + \ldots$
Then,

$$v_{rms} = \sqrt{\hat{v}^2 + \frac{\hat{v}_1}{\sqrt{2}}^2 + \frac{\hat{v}_2}{\sqrt{2}}^2 \ldots} = \sqrt{v_0^2 + \sum_{n=1}^{\infty}(\frac{V_m^2}{2})} \tag{3.13}$$

And if V_m is the rms value,

$$v_{rms} = \sqrt{v_0^2 + \sum_{n=1}^{\infty} V_m^2} \tag{3.14}$$

One thing to watch out for is that we could have a system with high voltage but near-zero current transferring little to no power. To account for this we look at the power factor 'PF'

Definition 8 **Power Factor**

$$PF \equiv \frac{\text{average power}}{\text{rms voltage} \cdot \text{rms current}} \tag{3.15}$$

For a sinusoidal V, I:

$$\begin{aligned} PF &= \frac{\frac{1}{2}\hat{V}\hat{I}\cos\phi}{\frac{\hat{V}}{\sqrt{2}} \cdot \frac{\hat{I}}{\sqrt{2}}} \\ &= \cos\phi \end{aligned} \tag{3.16}$$

If signals are not harmonics $PF = DF$. This can be a source of confusion.

For non-sinusoidal systems this becomes more difficult because, unlike pure signals, they may contain harmonics. In these systems either V, I, or both may contain harmonics. Generally in household power V is clean but I contains harmonics.

The effect of harmonics in currents is that $I_{\text{harmonics}}$ causes a higher I_{rms}; there is more current but no higher power! This reduces PF and causes $PF < DF$.

To summarize,

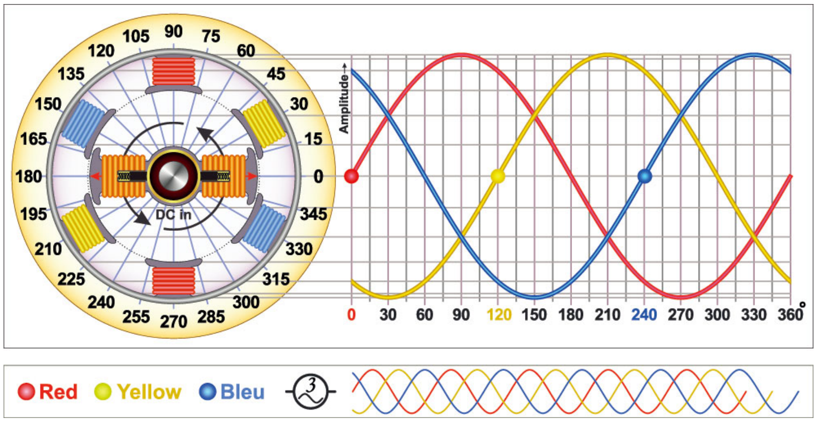
- In ideal systems, $PF = DF$ if all V and I are at one frequency.
- $PF = 1$ means no energy sloshing between load and source.

Harmonics are bad because they reduce the usefulness of the system There are very tight standards for how many harmonics one is allowed to inject into the system via generators or loads. See: textbook 16.6

Subsection 3.2

Lecture 5: Multi-Phase AC

AC power is generated by spinning a magnet between some coils. Some pixies get excited and by some Maxwell's equations and EMF and ECE259 we get a voltage induced in the coils.



Current from a generator with a single pair of coils is *single phase*. Most generators, like the one in the picture, have three pairs of coils and therefore generate three phases. For a typical three-phase setup with coils arranged at 0^o, 120^o, 240^o, the voltages can be found with a little bit of trigonometry.

$$\begin{aligned} v_a(t) &= \sqrt{2}V\cos\omega t \rightarrow \underline{V_a} = V\lfloor 0 \\ v_b(t) &= \sqrt{2}V\cos(\omega t - 120) \rightarrow \underline{V_b} = V\lfloor -120^o \\ v_c(t) &= \sqrt{2}V\cos(\omega t - 240) \rightarrow \underline{V_c} = V\lfloor 240^o \end{aligned} \tag{3.17}$$

And similar expressions may be derived for single-phase and two-phase power.

The reason why three-phase power is typically used has to do with efficiency of power transfer relative to the amount of wires and copper needed. Whereas single-phase power requires two wires to carry power and two-phase power requires four wires, three-phase power can be transmitted over 6, 4, or three wires. This saves a lot of copper as three-phase 3ϕ power can carry 50% more power than 2ϕ power for less copper.

In 3ϕ systems the voltages sum to zero; $v_a(t) + v_b(t) + v_c(t) = 0 \forall t$

Four-wire three-phase power:

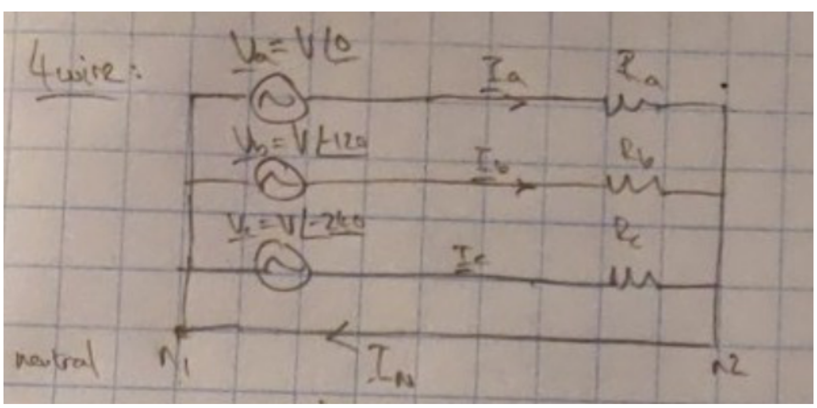


Figure 2. A four wire system for three phase power

If $R_a = R_b = R_c = R$, we have a balanced load and then the currents are related by $i_n = i_a(t) + i_b(t) + i_c(t) = 0$.

Proof: just substitute the condition earlier that voltages sum to zero and the fact that all resistances are equal into $I = \frac{V}{R}$

Three-wire systems drop the fourth neutral wire since it carries no current. This can be problematic if the load is not balanced; though $I_a + I_b + I_c = 0$ will still hold true since there is no return path, the nodes at either end of the AC source and resistor pair would have differing voltages.

If the system is balanced then to save ourselves from drawing everything out all the time, only a single diagram is drawn for a characteristic phase and then solved once.

Example

$$Z_a = Z_b = Z_c = Z \tag{3.18}$$

$$\underline{V_b V_a} e^{\frac{-j2n}{3}} \tag{3.19}$$

$$\underline{V_c V_a} e^{\frac{-j4n}{3}} \tag{3.20}$$

Then,

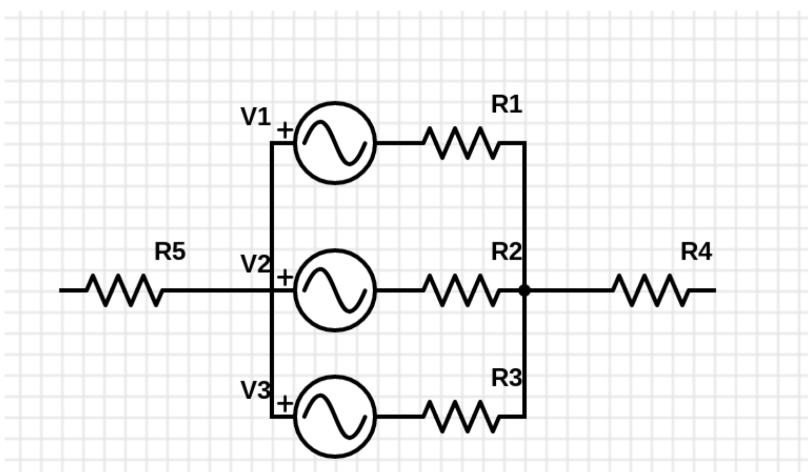
$$\underline{I_a} = \frac{\underline{\underline{V_a}}}{Z} \tag{3.21}$$

$$\underline{I_b} = \frac{\underline{\underline{V_b}}}{Z} = \frac{\underline{\underline{V_a}}}{Z} e^{\frac{-j2n}{3}} \tag{3.22}$$

$$\underline{I_b} = \frac{\underline{\underline{V_c}}}{Z} = \frac{\underline{\underline{V_a}}}{Z} e^{\frac{-j4n}{3}} \tag{3.23}$$

And then the solutions for phase b and c are the same as that for phase a except for the $120^o, 240^o$ offsets.

Because of this property, instead of drawing three separate diagrams for each phase, we can just draw one diagram and then solve for the other two phases.



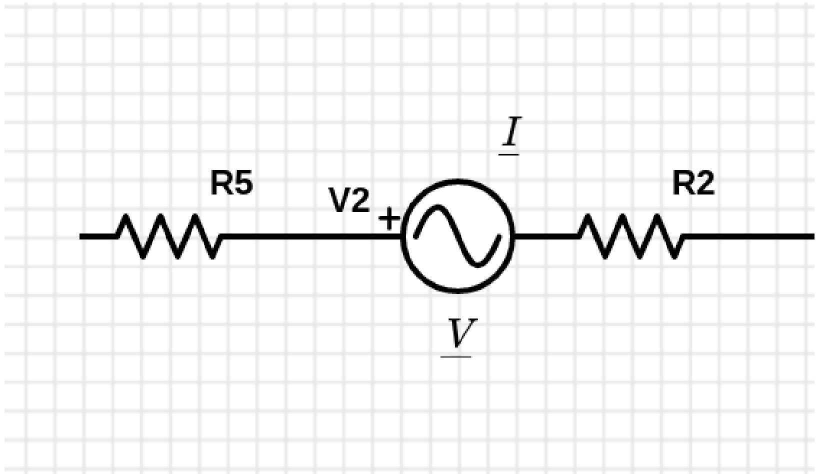


Figure 3. Condensed diagram for three phase power

Subsection 3.3
Lecture 6: Y and Delta connections

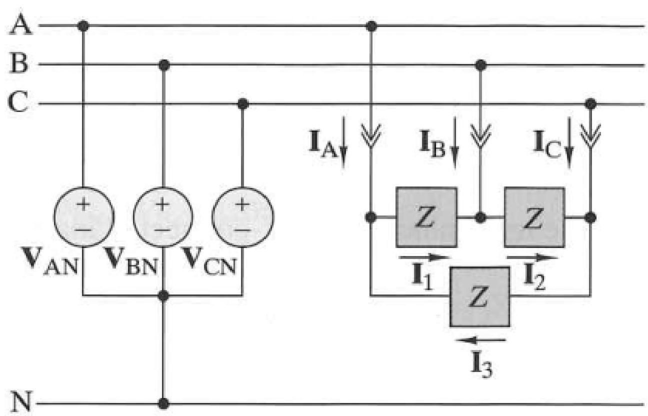
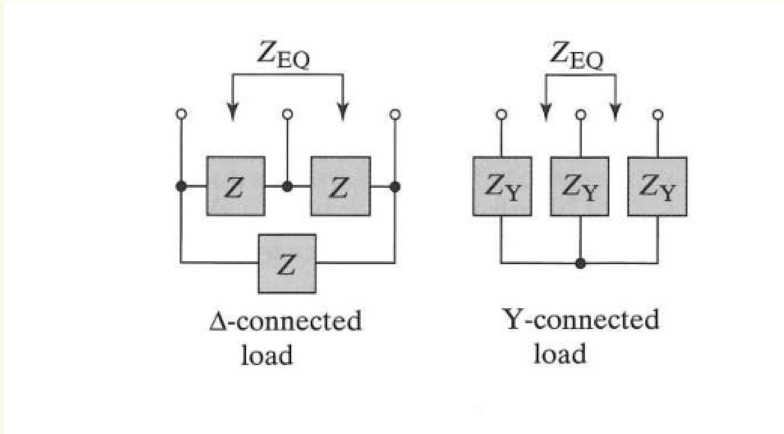


Figure 4. Three-phase system with a Y connected source and a Δ connected load

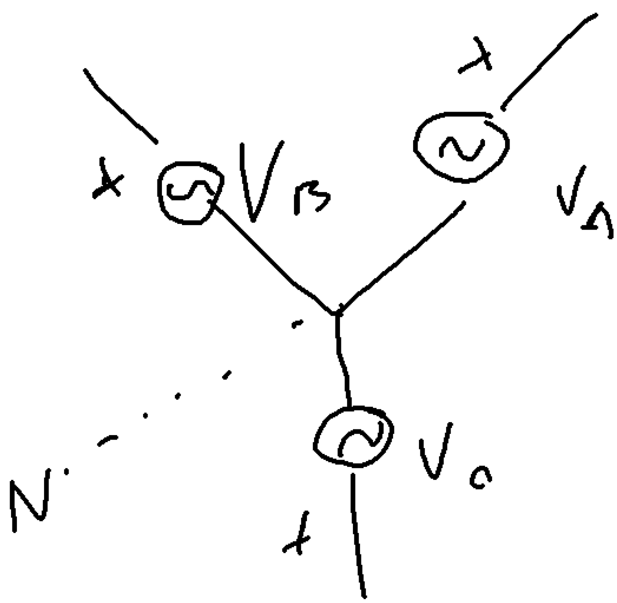
The current to ground I_N is always 0 for a Δ connected load. I_N is zero for a Y cnonnected load if Y has no neutral connection and the load/source are balanced.

Theorem 1 $Y - \Delta$ **conversion**
Any Δ load can be converted to a equivalent Y connected load



The derivation is in the textbook.
Sources can be connected in Y or Δ configurations.

Definition 9 Line currents are the currents on the lines a, b, c. Phase currents are the currents immediately beside the sources.

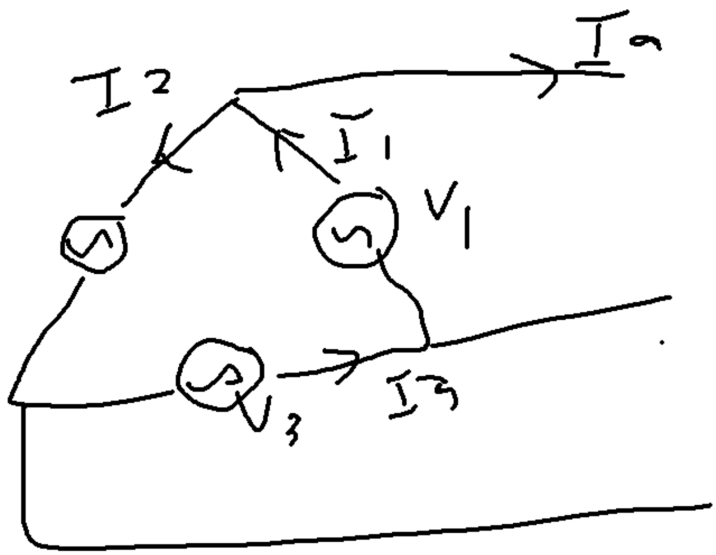


A Y connected source has the neutral line in the center.

$$\tilde{V}_{AB} = \tilde{V}_A - \tilde{V}_B = |\tilde{V}_A|(1 - 1\lfloor e^{-\frac{2\pi}{3}}) = \sqrt{3}|\tilde{V}_A|e^{\frac{j\pi}{6}} \tag{3.24}$$

Angle differences are $0, 120, 240$ degrees

The line and phase current are the exact same in the case of a Y connected source. However, the line and phase voltages are different and are related by (3.24).



A Δ connected source has no neutral point.

$$I_a = I_1 - I_2 = I_1(1 - 1\lfloor e^{-\frac{2\pi}{3}}) = \sqrt{3} I_1 e^{\frac{j\pi}{6}} \tag{3.25}$$

The line and phase voltages are the exact same in the case of a Δ connected source. However, the line current and phase currents are different and are related by (3.25).

Since neutral is not always available, we write 3ϕ systems based on their line-to-line voltages. For exampl,e, a $208V$ system has $120V_{rms}$ on each phase.

Example

A Y connected three-phase source has a line-to-line voltage, i.e. $V_{A \to B}$ of $208V$ but each source has $120V$ since $\frac{208}{\sqrt{3}} = 120V$. A similar argument can be applied for the Δ sources.

$$I = \frac{10}{\sqrt{3} = 5.8A} \tag{3.26}$$

Subsection 3.4

Lecture 7: DC-DC conversion

Let's say we want to supply $10V$ DC to a load but our supply is at 20.5V. We have a few options. We can use a resistor to drop the voltage via a voltage divide, which we learned about in ECE159. However this is not only not every efficient, but also non-responsive to changes in resistance load and will provide incorrect supply voltage if the load is not as anticipated. Another option is to use **transistors** to regular the power instead.

semiconductors to process power ⇔ power electronics

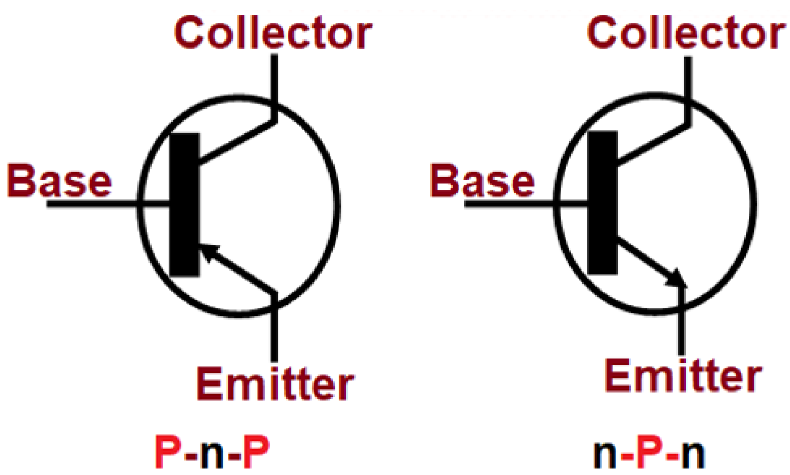


Figure 5. Transistors have a Base, Collector, and Emitter. At this point in the course, we just need to know that we can change the $V - I$ response by fiddling with the base current i_B

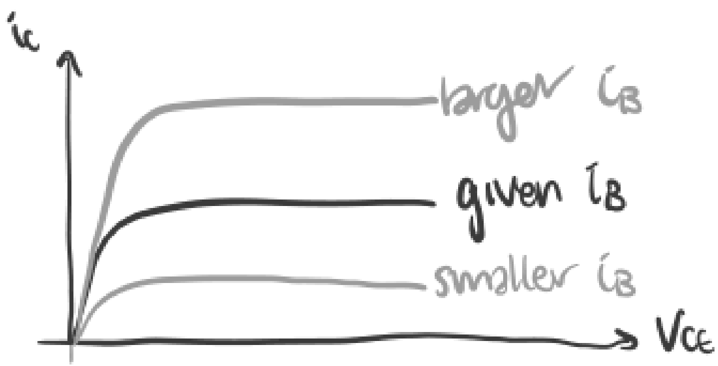
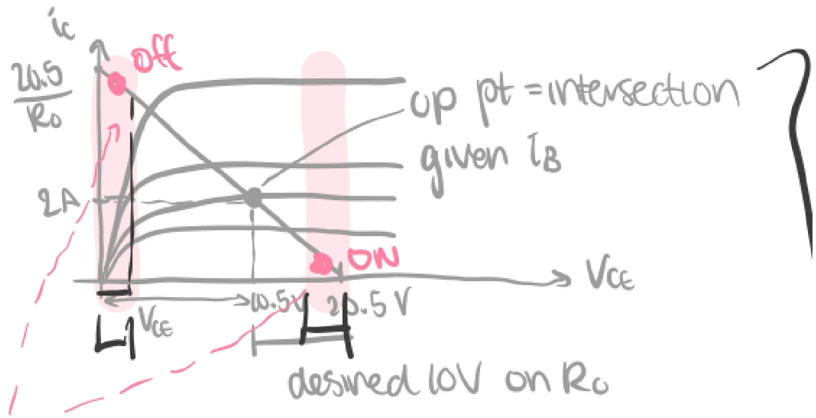


Figure 6. Thanks sherry for this figure

Finding the operating point that we want the transistor to be at can then be done graphically by solving a system of two equations with two unknowns, namely finding the intersection between the transistor response at at given i_B and the desired voltage/current of the load. Then the correct i_B can be selected.

However this is not very efficient since a lot of power is being wasted in the transistor. For a $20.5V$ source and a $10V$ load at $2A$, $V_{CE} = 10.5V$, so $P_{loss} = 10.5 * 2 = 21W$ being lost in the transistor which results in the rather poor $\frac{20}{20.5*2=41} \to \eta = 49\%$ efficiency. So the transistor is a little bit better but we still lose half the power – which means that in a device this would halve the battery life and require a huge heat sink. How can we do better?

Switch-mode power is a way to maximize η by switching between two states that correspond to low power loss at a high frequency



The left state is the 'off' state and the right state is the 'on' state. Switching rapidly between $20.5V, 0A$ and $0.5V, 4A$ results in what is, on which is on average what we want. This will also produce the same power output while increasing efficiency

What about the timing between the off/on states? This can be found by taking an integral to find the relationship between T_{on} and T_{off} with respect to power.

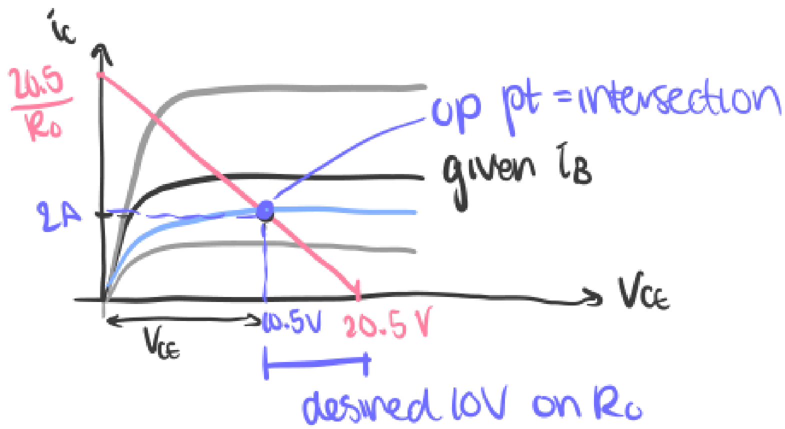
$$\begin{aligned} P &= \frac{1}{T}\int_0^T p(t)dt \\ &= \frac{1}{T}\int_0^{T_{on}} i_c^2 R_o dt \\ &= \frac{1}{T}\int_0^{T_{on}} 4^2 \cdot 5 dt \\ &= 80\frac{T_{on}}{T} \end{aligned} \tag{3.27}$$

So in order to transfer the same $20W$ to the 5Ω load, we would use $T_{on}/T = \frac{1}{4}$, which would correspond to $P_{in} = V_i \cdot T_{on}/T = 20.5W$. It then follows that $\eta = \frac{20}{20.5} = 97.6\%$ which is much better[1]

[1] *compared to the 49% before*

A few points of caution:

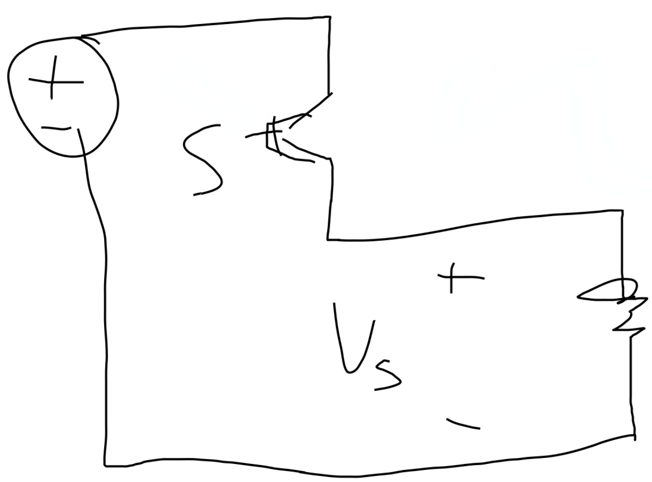
- A pulsing output voltage/power can be dangerous for some loads. For example motors aren't a fan of it. In these cases a filter can be applied to leave behind the average power.
- If we want to supply average voltage/power an on-time of 50% would be required.



Subsection 3.5

Lecture 8: Average output voltage

Consider a circuit with a transistor and a load.

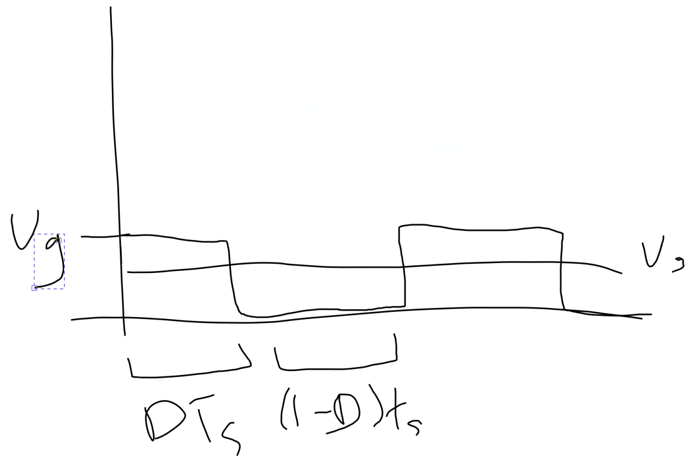


A switching signal S taking on a value from $0, 1$ periodically over time is applied to the transistor

Definition 10 **Duty cycle**: the fraction of time that S is 1

$$D = \frac{T_{on}}{T_s} \tag{3.28}$$

The response of the circuit, where V_g is the source voltage and V_s being the voltage at the load would look like this:



One problem that we want to characteristic is the ripple voltage, i.e. unwanted harmonics

$$v_s(t) = V_s + \sum_{k=1}^{\infty} V_s^k \sin\left(k \frac{2\pi}{T_s} t\right) \tag{3.29}$$

The V_s term is desirable, and the harmonic terms V_s^k are undesirable.

$$V_s^k = \frac{2V_g}{\pi} \sin(k\pi D) \tag{3.30}$$

Basically all Fourier analysis for power electronics will basically always result in an expression with a bunch of terms on one end and then a sin or cos. $\frac{1}{k}\frac{2V_g}{\pi}$ is the envelope of the harmonic terms. Note how the envelope decreases with k.

Plotting out the spectrum of signals we get:

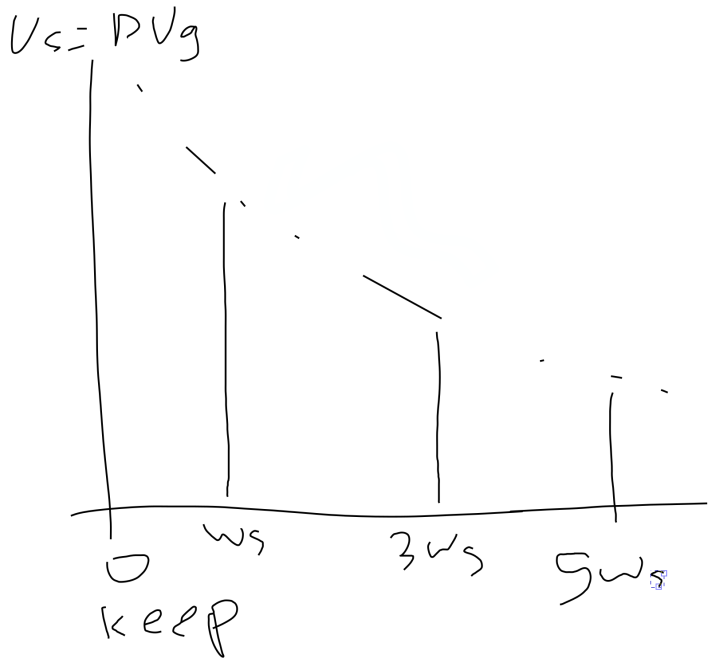
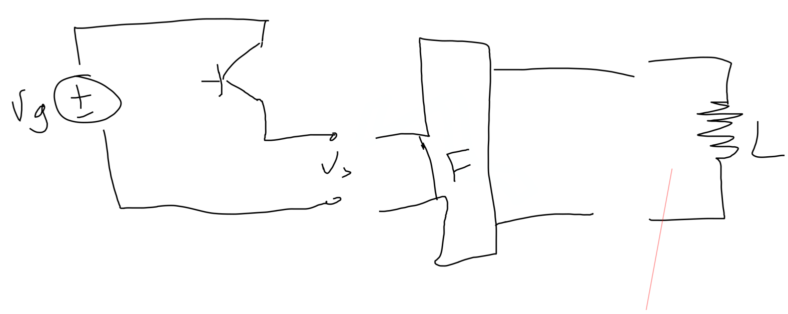


Figure 7. Voltage vs frequency. We want to keep the $\omega = 0$ term and then eliminate the rest with a filter. Note switching frequency $\omega_s = \frac{2\pi}{T_s}$

3.5.1 Filtering

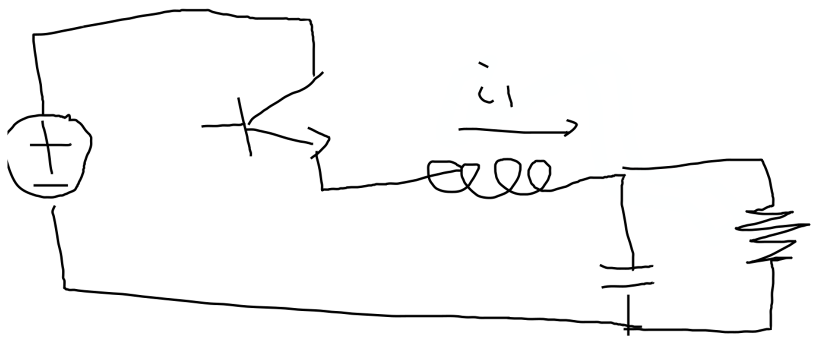
We'll need a "low pass" filter for dc/dc filtering.



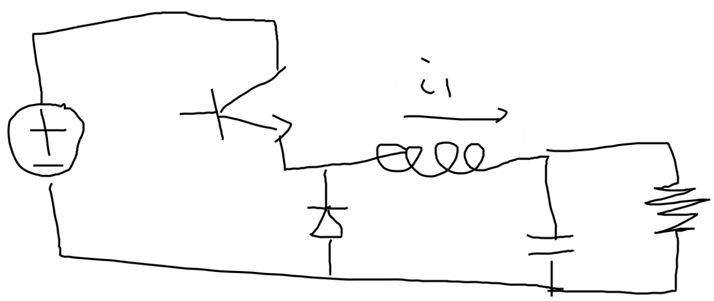
Choices for filters that may work for this system are the RC, LR , and LC filters. The LC is most promising because it is second order and it's theoretically lossless because there is no resistor.

We are going to want to maintain continuity of capacitor voltages and inductor currents. Things work out pretty well while we're on our duty cycle, but once the duty cycle turns off and the switch is opened, $i \to 0$. This is a problem because this will cause us to lose all the energy stored in the inductor everything the switch is opened, which can be a lot given the frequency that these devices operate at.

$$E = \frac{1}{2}Li^2 \tag{3.31}$$



In order to handle this, we must give the current somewhere to go while off-duty cycle; an energy-recovery diode which ensures that i_l is continuous and therefore E_l is lost.



Comment Switching should never yield discontinuity in i_l or v_c.

Some design considerations:

1. Filters all store energy during 'on' states and releasing it during 'off' states in the magnetic fields of inductors and electric fields of capacitors.
2. Faster switching $\leftrightarrow$ less stored energy needed $\leftrightarrow$ smaller capacitor and inductor needed
   - Switching too fast would cause the system to spend too much time during switching in undesirable high-loss states

Capacitors tend to be cheaper so designs are biased to store energy in them instead of inductors

Solving these DC-DC converters with Fourier analysis and superposition every time can be a pain, so we'll be looking at other methods next lecture. Plus, Fourier analysis requires an known $v(t)$ or $i(t)$ sources, which is not always the case. To motivate this, let's consider the case of an impulse being applied to the system or if the circuit contains diodes.

Periodic mapping using differential equations is a powerful method for solving transients and steady states from solving piecewise linear D.E.s. This is a whole ton of work so a third method was developed; small-ripple approximation. It only works for steady state[2] as it is an approximate solution to the differential equations based on periodic mapping

[2] *Well, we only use it for steady-state*

Subsection 3.6

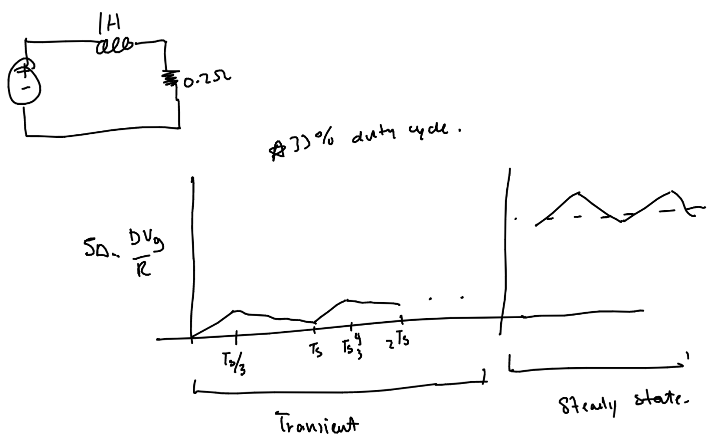
Lecture 9

A periodic system is in steady state if

1. all input sources are periodic
2. All states are periodic
3. switching events are periodic

Switching events being literally switching parts of the circuit on/off, i.e. with a transistor

Consider a simple LR circuit with a 33% duty cycle.



The system will have an initial transient response before settling down into a steady state. How can we find this steady state without having to solve for the transients until we get there?

Definition 11 **Steady state**: the net change of state variables over T_s is zero

An easier way to find the steady state is to use **state space averaging**.
Let's look at a RL circuit and its' steady-state response.

In power electronics this is usually termed small ripple assumption

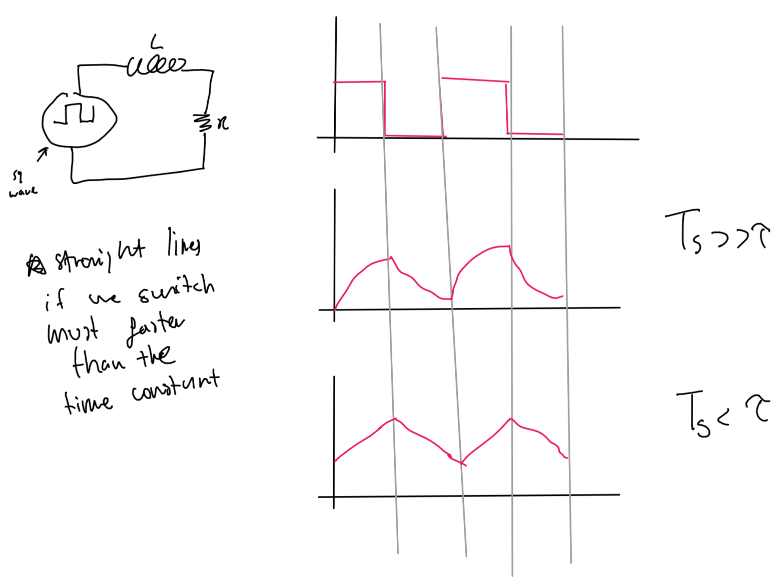


Figure 8. Note how the steady-state response becomes a bunch of straight lines if we switch faster than the time constant τ

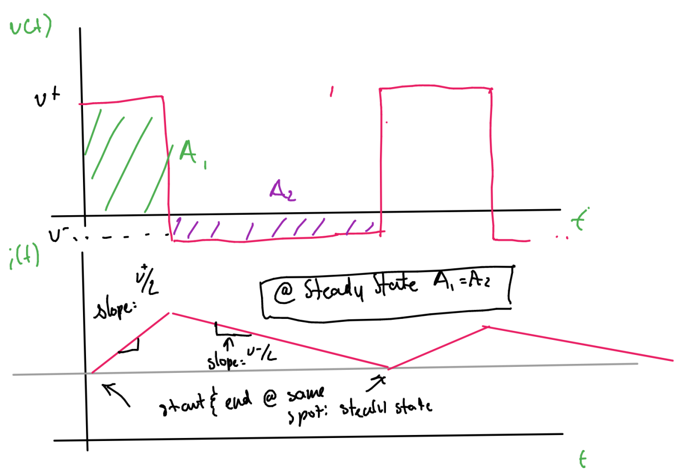
3.6.1 Inductor VoH-Seconds balance

$$\begin{aligned} v_l &= L\frac{di_l}{dt} \\ i_l(t) &= \frac{1}{L}\int v_l(t)dt \\ i_l(t+T_s) &= \frac{1}{L}\int_{t_0}^{t_0+T_s} v_l(t)dt + i_l(t_0) \\ \text{Note: } 0 &= \frac{1}{L}\int_{t_0}^{t_0+T_s} v_l(t)dt \end{aligned} \tag{3.32}$$

So, in steady state $i_l(t_0+T_s) = i_l(t_0)$

$$0 = \frac{1}{L}\int_{t_0}^{t_0+T_s} v_l(t)dt \tag{3.33}$$

This implies that, graphically, the area in the positive part of the $v(t)$ plot is equal to that of the negative part.



An analogous relationship can be found for capacitors.

3.6.2 Capacitor charge balance

$$\begin{aligned} i_c(t) &= \frac{dv_c(t)}{dt} \\ v_c(t_0+T_s) &= \frac{1}{C}\int_{t_0}^{t_0+T_s} i_c(t)dt + v_c(t) \end{aligned} \tag{3.34}$$

Capacitor balance gives current * time which is charge

And at steady state

$$0 = \frac{1}{C}\int_{t_0}^{t_0+T_s} i_c(t)dt \tag{3.35}$$

The relationship between the areas above/after the current curve and the voltage response returning to the same spots is the same as that for inductors, just flipped (i.e. voltage areas are equal for inductors but current areas are the same for capacitors).

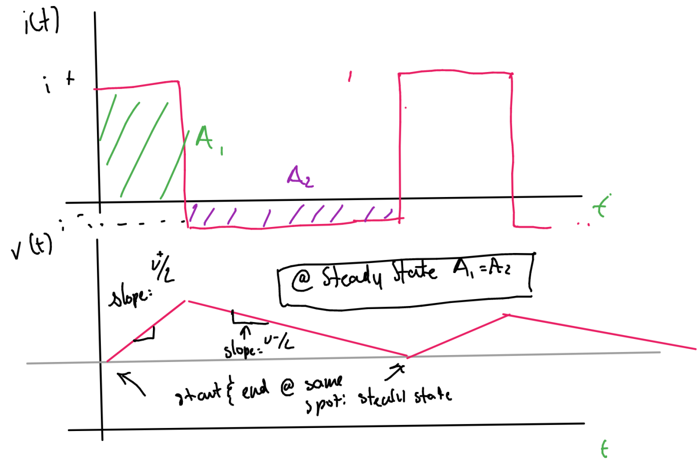


Figure 9. This $A_1 = A_2$ is the capacitor charge balance

As an aside it makes our lives easier to define the average current at steady state I_R so that instead of doing all the math relative to 0 we can take it to the center of the signal.

A_1 is adding charge to the capacitor, A_2 is removing charge

What we really want to find is the average v_c and i_l

Subsection 3.7
Lecture 10

1. Assume all switches are ideal
2. Apply small ripple assumption

$$
\begin{aligned}
i_l(t) &\approx I_l \\
v_l(t) &\approx V_c
\end{aligned}
\tag{3.36}
$$

3. Sketch waveforms for $v_l(t)$ for inductors and $i_c(t)$ for capacitors. These should generally be square in wave shape.
4. Apply V-s balance and charge balance
5. Validate ripple sizes and small current assumption

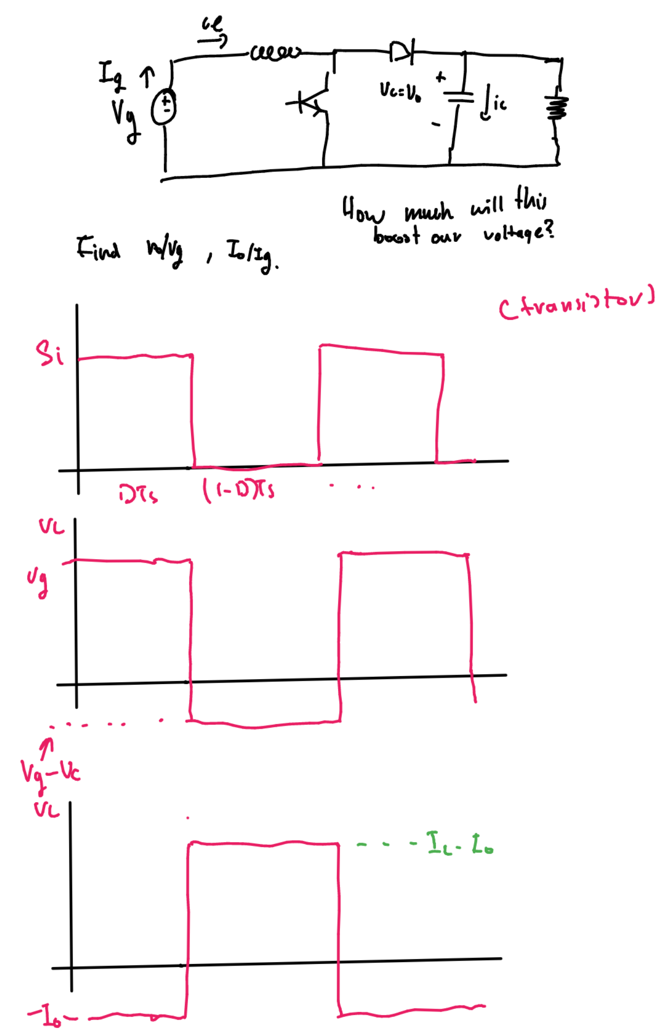


Figure 10. Example with a boost converter. Note: 3rd plot should be i_c vs t

Applying the $V - s$ balance:

$$\begin{aligned}\int_0^{T_s} v(t) &= 0 \qquad \text{or: } A_1 = A_2 \\ V_g D T_s + (V_g - V_c)(1-D)T_s &= 0 \\ V_g &= V_c(1-D) \\ \frac{V_o}{V_g} &= \frac{1}{1-D}\end{aligned} \tag{3.37}$$

What this means is that we can boost v strongly by modifying the duty cycle D. In theory we can do this infinitely far but in practice after a while physical effects tend to make it really difficult to boost further.

Applying a charge balance:

$$\int_0^{T_s} = i_c(t)dt$$

$$-I_o(DT_s) + (I_L - I_o)(1-D)T_s = 0 \tag{3.38}$$

$$\frac{I_o}{(I_l = I_g)} = 1 - D$$

These integrals can be found by looking at the graphs we drew with the square waves etc and the areas

So the output current can similarly be modulated via the duty cycle.
Let's check whether or not if the small ripple assumption is valid.

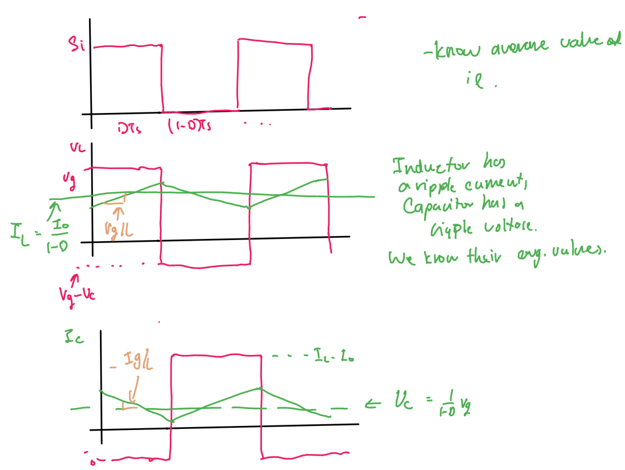


Figure 11. Inductor has a ripple current and the capacitor has a ripple voltage. The slope of the roughly saw-shaped waveform is inversely proportional to L, C. So we can pick L, C such that we get a small enough ripple.

So, what defines a 'small enough ripple?'
The amplitude of the ripple is defined from the peak to peak value of the waveform. [3]

[3] The textbook defines it as center to peak, but prof likes peak2peak

$$\Delta I_{lp2p} = \frac{1}{L}\int_0^{DT_s} V_g dt = \frac{V_g}{:} DT_s \tag{3.39}$$

$$\Delta V_{op2p} = |\frac{1}{C}\int_0^{DT_s} -I_o dt| = \frac{I_o}{c} DT_s \tag{3.40}$$

Defining a small ripple is a little more hand-wavy but usually indicates the point at which our approximations start to break down. Generally it means within $< 30\%$ of the rated current and $< 30\%$ of the rate voltage. But this is highly dependent on the application; a computer PSU would want something a lot cleaner for example.

$V_o = V_c$. Generally we don't care about I_{lp2p}; it is an internal variable. User only cares about V_o, not V_c – they're the same in this case but not necessarily.

3.7.1 Design Process

1. The application will tell the load; I_o, V_o
2. We will have to pick the switch. It'll have to meet the current spec and maybe some thermal design
3. Look at switch data sheet and then look at the switching frequency that it can run at[4]
4. Compute required duty cycle D for the desired output characteristics V_o/V_g

[4] *Usually smaller switches can run faster than larger ones*

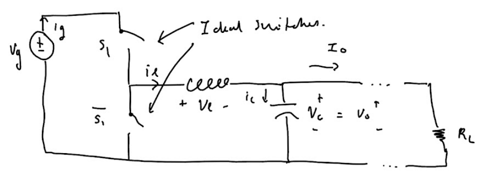
5. Pick inductor current to achieve the small ripple assumption to get reasonable ΔI_{lp2p}. Inductors are more expensive than capacitors so most try to get it to within $< 20\%$ of rated and do the rest using capacitors

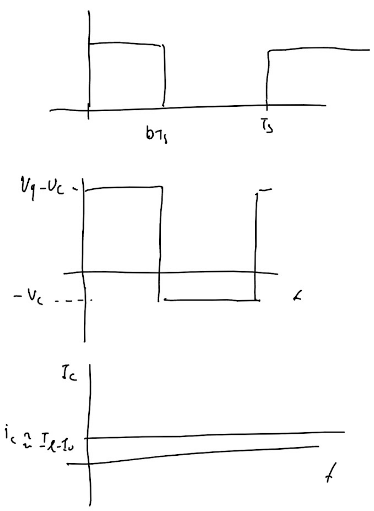
6. Pick C to meet ΔV_{cp2p} specification

Subsection 3.8

Lecture 11

I slept in and was behind on notes so here are handwritten ones.





V-S Balance

$$\int_0^{T_s} V_L \, dt = 0$$

$$(V_g - V_c) D T_s - (1-D) T_s V_c = 0$$

$$V_c = D V_g$$

$$\frac{V_o}{V_g} = D$$

$$\int_0^{T_s} i_c \, dt = 0$$

$$\int_0^{T_s} (I_L - I_o) \, dt = 0$$

$$I_L - I_o = 0, \quad I_c = 0.$$

$$\frac{I_o}{I_g} =$$

→ Take average.

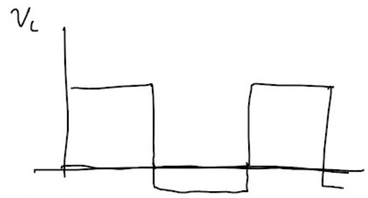
$$I_g = \frac{1}{T_s} \int_0^{T_s} i_g \, dt \Rightarrow I_g = D I_L$$

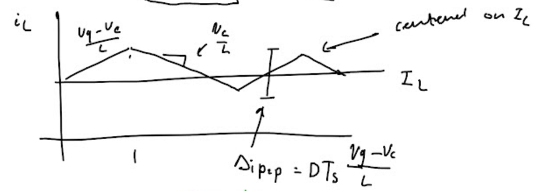
$$\frac{I_o}{I_g} = \frac{1}{D}$$

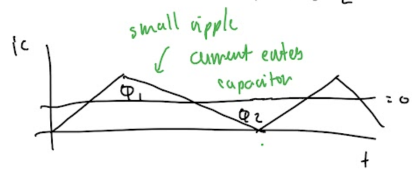
★ $I_o - I_L = 0$

$\therefore I_o = I_L$.

Steady ripple to size L, C

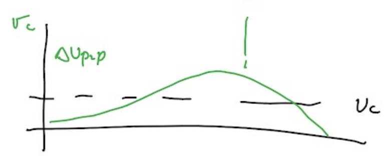






We previously treated i_c as a straight line. How can we approximate the value of the capacitor current i_c?

The difference between i_o, i_c should actually cause a small ripple in real world currents.



$$\boxed{\Delta i_{L,p\text{-}p} = D T_s \frac{V_g - V_o}{L}}$$ Design eq #1

$$\Delta V_{p\text{-}p} = \frac{Q_1}{C} =$$

$$Q_1 = \frac{1}{2}\,\frac{T_s}{2}\,\frac{\Delta i_{L,p\text{-}p}}{2}$$

$$= \frac{T_s}{8}\, D T_s \frac{V_g - V_o}{L}$$

$$\boxed{\Delta V_{p\text{-}p} = \frac{T_s^2}{8} D \frac{V_g - V_o}{LC}}$$ Design eq #2

$$V_o = D V_g \quad \therefore V_g = \frac{1}{D} V_o$$

$$(V_g - V_o) = \left(\frac{1}{D} - 1\right) V_o$$

$$= \frac{1}{D}(1 - D) V_o$$

can rewrite in terms of others...

$$\boxed{\frac{\Delta V_{p\text{-}p}}{V_o} = \frac{T_s^2}{8}\,\frac{(1-D)}{LC}}$$ Alternative design eq

Subsection 3.9
Lecture 12

ECE352: Computer Organization

PART II

Taught by Prof. Andreas Moshovos

Section 4
Admin stuff

Subsection 4.1
Lecture 1

- Lecture recordings on YouTube
- Online notes: https://www.eecg.utoronto.ca/ moshovos/ECE352-2022/
- Course will cover the following:
  - C to assembly
  - How to build a processor that works
  - Intro to processor optimizations
  - Peripherals
  - OS support (Maybe)
  - (Maybe) Arithmetic circuits
  - Use NIOS II and cover a little bit of RISC-V

4.1.1 Mark breakdown

- Labs 15%
- project 5%
- midterm 30%
- Final 50%
- All exams will be open notes/book/whatever except another person/service helping you.

Section 5
Preliminary

Subsection 5.1
Lecture 2: Using binary quantities to represent other things

Computers can represent information in bits; 0/1. Though they don't necessarily know or care what bits are, we may assign our own arbitrary meaning to them – usually numbers with the help of positioning; the LSB represents 2^0 and so forth.

C types

Or, just `;;include <stdint.h>`...

- int: 32b (word)

- char: 8b (byte)
- short: 16b (half word)
- long: 32b (word)
- long long: 64b

Signed numbers may be represented in a number of ways.

- Sign bit (make MSB represent positive or negative numbers and then the remaining $n-1$ bits represent the number. Con: hardware impl sucks because requires if/else
- Two's complement [5]. Pro: only need to implement adders on hardware and then negative numbers will work just like any other except must be interpreted differently. Positive numbers would always start with a 0 and negatives would start with 1. So the range of possible values becomes $-(2^{n-1}-1), +2^{n-1}-1$

[5] Flip bits, add one. Intuition; in 3 bit system, adding 7 to 1 would result in 8 which would get truncated to 0.

Adding together binary numbers can also cause overflow; $(A+B) \geq A, (A+B) \geq B$ may not always be true. Also, when we work with these types we always use all the bits. This has implications when working with values of different lengths.

- `char b = -1` (1111 1111))
- `short int c = -1` (0000 0000 0000 0001))
- `a = b + c` 0000 0001 0000 0000
- In order to deal with this we must `cast` the `char` to a `short int`. This is done via sign extension which prepends 0s or 1s [6] to the `char` so that math can be done on it.

[6] two's complement

5.1.1 Floating Point Numbers

Whereas fixed point numbers i.e. \$5.25 can be represented just as how an integer would be represented but with the understanding that the user would interpret it as having a decimal point somewhere that indicates the position of 2^0. This decimal point would be the same for all numbers of that type, i.e. we could have a six bit number that has places $2^2 2^1 2^0 2^{-1} 2^{-2}$ This is common in embedded systems and how it is formatted isn't super clearly standardized.

Lemma 1 Reference: What Every Computer Scientist Should Know About Floats

Definition 12 **IEEE 754 Floating Point**
This is a single precision 32 bit float

```
S EEEEEEEE MMMM MMMM MMMM MMMM MMMM MMM
```
(5.1)

The most significant `S` bit is the sign bit, bits 30 through 23 `E` form the exponent which is an unsigned integer, and 22 through 0 form the (`M`)antissa. The number being represented can be found using the following:

$$(-1)^S \times 2^{(E-127)} \times 1.\text{Mantissa} \tag{5.2}$$

Example

> For example, given the following float:
>
> $$1 \quad 10000001 \quad 10000000000000000000000$$
>
> So S = 1, E = 10000001 = 129 and Mantissa = 10000000000000000000000. The number is therefore
>
> $$(-1^1) \times 2^{(129-127)} \times 1.10000000000000000000000 = -6.0 \tag{5.3}$$

IEEE754 also defines 64 bit floating-point numbers. They behave the same except for now having an 11 bit exponent, the bias being 2047[7], and the mantissa having 52 bits.
A few special cases are also available to represent other quantities

- If E=0, M non-zero, value=$(-1)^S \times 2^{-126} \times 0.M$ (denormals)
- If E=0, M zero and S=1, value=-0
- If E=0, M zero and S=0, value=0
- If E=1...1, M non-zero, value=NaN 'not a number'
- If E=1...1, M zero and S=1, value=-infinity
- If E=1...1, M zero and S=0, value=infinity

[7] *Instead of 127*

In C, `float` is a 32 bit float and `double` is 64

Floating-point numbers are inherently imprecise. Addition and subtract are inherently lossy; the mantissa window may not be large enough to capture the decimal points. Multiplication and division just creates a ton of numbers.
Converting real numbers to IEEE754 floats, here using 37.64 as an example, can be done as follows

- Repeatedly divide the part of the number > 0 by 2 and get the remainders, i.e. 37/2 = 18, rem = 1 -> 18/2 = 0, rem = 0 -> 4/ 2 = 2, rem = 0, 2/2 = 1, rem = 0, 1/2 = 0, rem = 1. As a 2 bit number E is 100101. But we need to convert it to IEEE754 format with the exponent; E - 127 = 5, E = 132 = 1000 0100.
- Do the same for the part of the number past the decimal, but multiplying by two and checking if > 1: 0.64 * 2 = 1.28 -> 1, 0.28 * 2 = 0.56 -> 0, 0.56 * 2 = 1.12 -> 1 ... and so forth. At some point we will hit a cycle but we'll just take the N_{mantissa} of digits.

So the full number is 01000010000101101000111101011111

There are more floating point formats introduced by nvidia and google such as a half-precision or 8-bit float designed to reduce memory use for machine learning

Section 6

NIOS II Preliminary

Subsection 6.1

Lecture 3: Behavioural Model of Memory

Computers can be described as a set of units, each of which interact with each other and the outside world in a specified way. For example, modern computers tend to have memory units, processing units, display units, and so forth. Each unit has a set of inputs and outputs, and a set of rules that govern how the unit behaves. This gives the manufacturer flexibility in how they want to implement a unit, as long as the unit behaves as specified. When designing these operational units it is important to strike a balance between functionality and specificity; if the unit is too specific it will be difficult to implement, but if it is too general it will be difficult to use.

Specification is the description of what an unit should do, and **implementation** is how it actually does it. For example, an `OR` gate can be specified as a truth table and then implemented via transistors or a person in a box.

6.1.1 Memory

Memory is a unit that stores information and is usually represented as a vector of elements, usually a byte (8 bites). Each element, or memory location, contains a binary quantity and has an associated *address*. The address is a number that uniquely identifies the location of the element in the vector, and is **permanently fixed** at time of manufacture. Most systems are byte-addressable, meaning that there is an unique address for each byte in the memory. The collection of all addresses is called the **address space** of the memory, which is typically a power of two. Modern systems tend to be 32 or 64 bit, meaning that the address space is 2^{32} or 2^{64} elements long.

For each memory location there there are two operations available

- **Read**: Read the value stored at a given address
- **Write**: Write a value to a given address

Typical memory behaviour models define that the order of memory operations matters, i.e.

1. `store 0x10, 0xf0`
2. `store 0x20, 0xf0`

We would see that `0xf0` contains the value `0x20` and not `0x10` due to the sequential execution model. Memory that adheres to the sequential model offers operations that are **atomic**; the operations are performed on its own with no interaction or overlap with anything else.

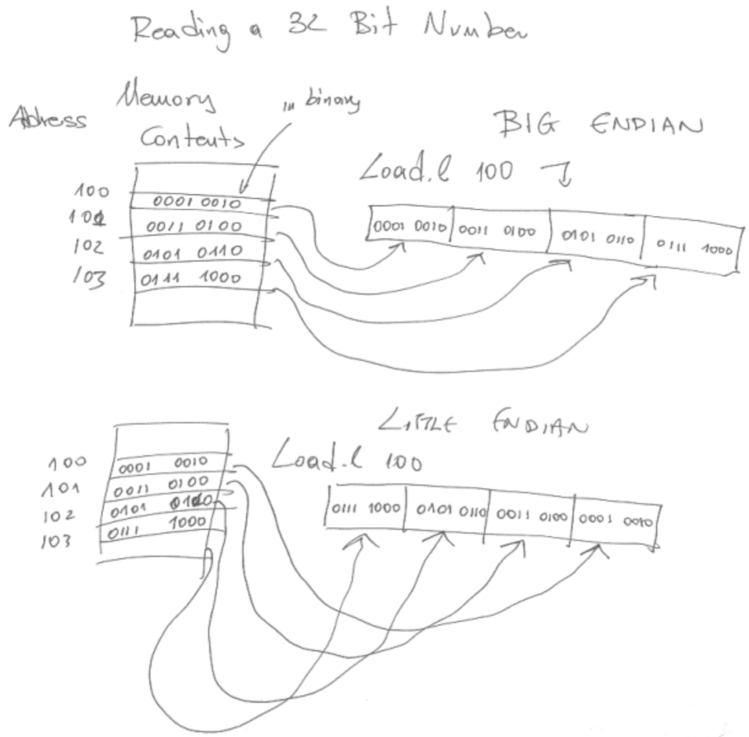
`In this case it is convenient to draw memory as an array where each row comprises four consecutive bytes.`

0x00 0000	0x11	0x22	0x33	0x44
0x00 0004	0xff	0x88	0x62	0x51
. . .				
0xff ffff				

Systems are generally also addressable by words, halfwords, and bytes. Different architectures have different constraints on allowing unaligned access[8]

Endianness refers to the order in which bytes are stored in memory. Though some processors are big-endian, most modern processors are little-endian. The NIOS II used for this course is little-endian.

[8] Aligned access only means to allow [only] reads or writes for a data size i.e. halfword to an address divisible by the size of said data type. For example an longword access on our development board would be at an address divisible by 4



6.1.2 Physical Interface

What physical interfaces would be necessary to implement this behavioural model?

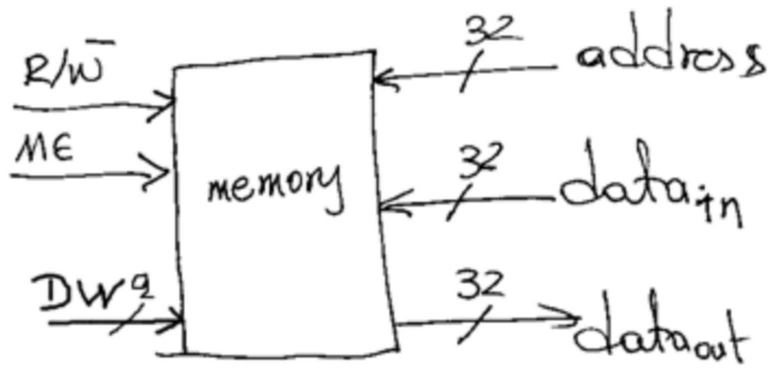
Given a summary of requirements as follows:

1. Read and write operations
2. Addressable by byte, word, longword
3. 24 bit address
4. 32 bit for writing
5. 32 bit for reading
6. signal for do nothing

A single bit signal can be used to indicate whether the memory is reading or writing, and a two bit signal can be used to specify if we're interested in addressing by byte, word, or longword. The address is 24 bits, so we need 24 address lines. As for reading/writing data, we have the option of having two 32 bit data lines, or multiplexing a single 32 bit line. A single bit signal can be used to indicate to do nothing or not.

The use of a single bit signal to indicate 'do nothing' is necessary because a physical device won't be able to change all signals instantaneously, so we use it to tell the memory to wait until these transient effects die off

One way of multiplexing the data lines is to use a tri-state buffer, which is a buffer that can be enabled or disabled. When enabled, the buffer acts as a normal buffer, but when disabled, the output is disconnected from the input. On the other hand this means that our memory chip would not support simultaneous reads or writes.



Subsection 6.2

Lecture 4: NIOS II Programming Model

The NIOS II assumes a `32-bit` address space where each address holds a single byte. Each byte is addressable, and three data types are supported. Halfword and word accesses must be aligned.

- **Byte**: 8 bits
- **Halfword**: 16 bits
- **Word**: 32 bits

The NIOS II also has a set of registers

- 32 general purpose 32 bit registers
  - `r0` is always zero
- 6 control registers, 32 bits each
- Program counter (PC), 32 bits

Many operations can be synthesized using another operation involving zero, i.e. assignment `A=B` can be implemented as `A = B + 0`

There are certain conventions for the use of registers, which are as follows:

Register	Name	Function	Register	Name	Function
r0	zero	0x00000000	r16		
r1	at	Assembler Temporary	r17		
r2		Return Value	r18		
r3		Return Value	r19		
r4		Register Arguments	r20		
r5		Register Arguments	r21		
r6		Register Arguments	r22		
r7		Register Arguments	r23		
r8		Caller-Saved Register	r24	et	Exception Temporary
r9		Caller-Saved Register	r25	bt	Breakpoint Temporary (1)
r10		Caller-Saved Register	r26	gp	Global Pointer
r11		Caller-Saved Register	r27	sp	Stack Pointer
r12		Caller-Saved Register	r28	fp	Frame Pointer
r13		Caller-Saved Register	r29	ea	Exception Return Address
r14		Caller-Saved Register	r30	ba	Breakpoint Return Address (1)
r15		Caller-Saved Register	r31	ra	Return Address

Notes to Table 3–1:
(1) This register is used exclusively by the JTAG debug module.

6.2.1 Adding Two Numbers

As an exercise, let's see how we can implement the following piece of code in NIOS II assembly

```
unsigned int a = 0x00000000;
unsigned int b = 0x00000001;
unsigned int c = 0x00000002;

a = b + c;
```

Register-only version

```
addi r9, r0, 0x1
addi r10, r0, 0x2
add r9, r10, r11
```

`addi` stands for 'add intermediate', the only difference being that the second operand is a number. It is used to set a constant

In general, most instructions take the form of `operation destination, source1, source2`.

Breaking it down even further we can see that these assembly instructions actually perform a number of steps

```
addi r9, r0, 0x1
        ; 1. read r0
        ; 2. Add value read in step 1 with 0x1
        ; 3. Write result of step 2 to r9
        ; 4. increment PC to next instruction
addi r10, r0, 0x2
        ; 1. read r0
        ; 2. Add value read in step 1 with 0x2
        ; 3. Write result of step 2 to r10
        ; 4. increment PC to next instruction
add r9, r10, r11
        ; 1. read r10
        ; 2. read r11
        ; 3. Add values read in steps 1 and 2
        ; 4. Write result of step 3 to r8
        ; 5. increment PC to next instruction
```

What about 32 bit constants? An unfortunate quirk is that `addi` only supports 16 bit constants, so we need to use `ori` to set the upper 16 bits of the register.

```
movhi r9, 0x1122
        ; Sets the upper 16 bits of r9 to 0x1122
        ; and the lower 16 bits to zero
ori r9, r9, 0x3344
        ; bitwise OR the value in r9 with 0x3344
        ; which will set the lower 16 bits to 0x3344
```

This is a PITA so NIOS II offers a few pseudo-instructions to make this easier

```
movi rX, Imm16
        ; sets rX to the sign-extended (signed) 16 bit immediate
movui rX, Imm16
        ; sets rX to a zero-extended unsigned 16 bit immediate
movia rX, Imm32
        ; sets rX to a 32 bit immediate
```

- Footnote1: `movia` does not use the `movhi` and `ori` instructions to create a 32-bit immediate but rather a `movhi` and a `addia`. `addi` will sign extend it's 16-bit field so some adjustment might be needed for whatever is being passed to `movhi`.
- Footnote2: movhi r9, %hi(0x11223344) is equivalent to movhi r9, 0x1122. Ori r9, %lo(0x11223344) is equivalent to ori 9, 0x3344. That is, %hi(Imm32) returns the upper 16-bits of Imm32 and %lo(Imm32) the lower 16 bits.
- Footnote3: movhi r9, %hiadj(0x11223344) followed by addi r1, %lo(0x11223344) is the correct way of creating a 32-bit immediate using movhi and addi. %hiadj(Imm32) returns the upper 16 bits of the immediate as-is or incremented by 1 if bit 15 is 1. Think why this is necessary based on footnote 1.
- Footnote4: %hi(), %lo(), and %hiadj() are macros supported by the assembler. They are not NIOS II instructions. They get parsed during compile time.

6.2.2 Adding two numbers using memory

NIOS II is a load/store architecture which means that all data manipulation happens only in registers.

```
; read b from memory into r9
movhi r11, 0x0020
ori   r11, r11, 0x0004
ldw   r9, 0x0(r11)

; read c from memory into r10
movhi r11, 0x0020
ori   r11, 0x0008
ldw   r10, 0x0(r11)

; add, then store into r8
add   r8, r9, r10

; store r8 into memory
movhi r11, 0x0020
ori   r11, r11, 0x0000
stw   r8, 0x0(r11)
```

The new instructions introduced here are

```
ldw rX, Imm16(rY) ;; 'load word' from memory
;; rX, rY registers, Imm16 is a 16 bit immediate
;; TLDR; Rx = mem[rY + sign-extended(Imm16)]
; 1. read rY
; 2. sign-extend Imm16 to 32bits
; 3. adds the result of step 1 and 2
; 4. reads from memory a word (32 bit) using the result of step 3 as the address
; 5. write the result of step 4 to rX
```

```
stw rX, Imm16(rY) ;; 'store word' to memory
;; rX, rY registers, Imm16 is a 16 bit immediate
;; TLDR; mem[rY + sign-extended(Imm16)] = rX
; 1. read rY
; 2. sign-extend Imm16 to 32bits
; 3. adds the result of step 1 and 2
; 4. write to memory rX using the result of step 3 as the address
```

This can be simplified using the `movia` macro

```
movia r11, 0x200004
ldw   r9, 0x0(r11)
movia r11, 0x200008
ldw   r10, 0x0(r11)
add   r8, r9, r10
movia r11, 0x200000
stw   r8, 0x0(r11)
```

In this lecture so far we have seen three addressing modes

1. Register addressing, i.e. `rX`
2. Immediate addressing, i.e. `Imm16`
3. Register indirect addressing with displacement, i.e. `Imm16(rY)`. This is how we calculate the referenced memory address. Register indirect refers to using a register's value to refer to memory, and 'displacement' refers to adding a constant prior to using the register value to access memory. Register indirect addressing is where we use a displacement of 0.

We can exploit register indirect addressing with displacement.

```
movhi r11, 0x0020
ori   r11, r11, 0x0004
ldw   r9, 0x0(r11)
;; can be replaced with
movhi r11, 0x0020
ldw r9, 0x4(r11)
```

Note that the value of `r11` does not change since the subsequent operations use an offset to that value.

Generally when we want to read memory from `A` we can use

```
movhi r11, (upper 16 bits of A)
ori   r9, r11, (lower 16 bits of A)
```

Care must be taken when the 16th bit of `A` is 1 since the addition that `ldw` performs will sign extend it to be a negative number, i.e.

```
movhi r11, 0x0020
ldw r9, 0x8000(r11)
;; this is incorrect because
;; will extend to 0xFFFF8000, which would result
;; in a final address of 0x001F800
```

This is where the macros `%hiadj(Imm32)` and `%lo(Imm32)` come in handy, since they will add 1 to the values if bit 15 of Imm32 is 1. This results in code that looks like this:

```
movhi r11, %hiadj(0x208000)
ldw r9, %lo(0x2080000)(r11)
;; will extend to 0xFFFF8000, which would result
;; in a final address of 0x001F800
```

Section 7

Assembly Basics

Subsection 7.1

Lecture 5: Simple Control Flow

We have prior worked with straight-line sequences. In this lecture we will look at how to add control flow to our programs, i.e if-then-else, etc.

A pseudo-c program will be rewritten in assembly to illustrate the concepts.

```
unsigned int a = 0x00000000;
unsigned int b = 0x11223344;
unsigned int c = 0x22334455;

if (b == 0)
then a = b + c;
else a = b - c;

```

The `data` section contains stuff that you want to be initialized for you before the entry point of the program is called, e.g. global variables. This segment as a fixed size. The `text` or `code` segment contains executable instructions (typically read-only, unless the architecture allows self-modifying code) and typically resides in the lower parts of memory. `bss` contains static and global variables which are zero-initialized; usually used for uninitialized data

```
      .section .data
va:   .long 0x0
vb:   .long 0x11223344
vc:   .long 0x55667788


main:
      movia r11, va
      ldw   r9, 4(r11)
      beq   r9, r0, then
else:
      ldw   r10, 8(r11)
      sub   r8, r9, r10
      stw   r8, 0(r11)
      beq   r0, r0, after

then:
      ldw   r10, 8(r11)
      add   r8, r9, r10
      stwio r8, 0(r11)
after:
```

Definition 13 We encounter two new instructions in this snippet.

- `sub` is a subtraction instruction
- `beq`: a branch-if-equals instruction

The `beq` instruction takes the general form
`beq RX, rY, label`.
This instruction will compare the values of rX and rY and if the condition is true then the program counter will jump to the destination label. When the branch changes the program counter it is called a taken branch, otherwise it is non-taken. Non-taken branches fall through to the next instructions.

Comment Note: whereas the assembly `beq` command is written as a comparison with a label to jump to, in the NIOSII instruction the destination is encoded relative to the instruction location with a 16-bit displacement constant. The displacement is therefore calculated as $PC + 4 +$ displacement, meaning that we can jump at most $+32774, -32772$ bytes[9] from the current program counter. The compiler will complain if the branch cannot be implemented.
Encoding is as follows:
`AAAAA BBBBB IIIIIIIIIIIIIIII 0x26`.
where A, B are register names (hence 5 bit), I is a 16 bit immediate value, and $0x26$ is the branch type, in this case beq
Most branches tend to be branch backwards because of loops.

[9] *16 bit signed constant+4, with 4-byte alignment since instructions are 4 bytes long.*

Other branch instructions include

- `br`: always/unconditional branch
- `bne`: branch if not equal
- `blt`: branch if less than, w/ signed comparison
- `bltu`: branch if less than, w/ unsigned comparison
- `bgt`: branch if greater than, w/ signed comparison
- `bgtu`: branch if greater than, w/ unsigned comparison

```
        .data
        .align 4 ;; Align to word size addresses which are faster to access
a: .word 0
b: .word 0x11223344
c: .word 0x55667788
        .text
        movia r11, a  ;; moves the address of a into r11
        ldw r9, 4(r11) ;; loads the value at address a+4 into r9
        ldw r10, 8(r11)
        add r8, r9, r10
        stw r8, 0(r11)
```

Subsection 7.2
Lecture 6, 7: For loops and arrays

How can we implement the following in assembly?

```
short arr[5] = { 1, 2, 3, 4, 5 }; // an array of word values
↪  (16 bit)
short n = 5;              // the number of elements in the array
short sum = 0;

for (i = 0; i  < n; i++)
   sum = sum + arr[i];
```

Arrays are typically implemented at the machine level as a contiguous block of memory.

Generally element `a[i]` is at address `&a[0] + sizeof(TYPE) * i`

```
                .data
                .align 1
arr .hword 1,2,3,4,5
n   .hword 5
sum .hword 0
```

Listing 1: Creating a static array in assembly

Multidimensional arrays are handled by having an array of pointers to arrays and dereferencing twice to arrive at the desired memory location. Alternatively we can allocate a $1x(m{\cdot}n)$ chunk of memory and then address it as $a[i][j] = a[i \cdot n + j]$.

As for the for loop, let's look at at what `c` does first.

`c` and pascal and other modern programming languages store arrays in row-major order, i.e. consecutive elements in a row are place together in memory. FORTRAN uses column-major.

```
for (init; cond; post)
        body
```

Listing 2: General form of a `c` for loop

Breaking it down a little more into atomic steps

```
INIT
if (!COND), we are done
BODY
POST
GOTO line 2
```

Listing 3: General form of a `c` for loop

Let's now rewrite this in assembly

```
                .text
forloop:
        add r8, r0, r0 ;; INIT
        movia r9, n ;; set COND, i.e. the n that we compare i to. See: loop invarient :)
        ldh r9, 0(r9)
;; movhi r9, %hiadj(n)
;; ldh r9, %lo(n)(r9) ;; this code block is a shorter piece with the same effect
loop:
        bge r8, r9, endloop ;; test condition for when we are done
        ;; body goes here
        ;; let's use r10 to store the running sum at in the end we can load it to memory
        movia r11, arr ;; load the address of arr[0] into r11
        ;; add the index to the address of arr[0]; &arr + i
        add r11, r11, r8
        ;; add again; &arr + i + i = &arr + 2i
        add r11, r11, r8
        ldhio r12, 0(r11) ;; r12 = arr[i]
        add r10, r10, r12 ;; r[10] += arr[i]

        addi r8, r8, 1 ;; POST i.e. r8 += 1
        br loop
endloop:
        movia r11, sum
        sth r12, 0(r11) ;; write the sum to memory
```

A similar approach can be taken for while loops.

```
.text

add       r8, r0, r0 ;; zero out r8
movia     r9, n ;; assumes n >= 1
ldh       r9, 0(r9) ;; loads condition

doloop:
                                        movia r11, arr ;; grab address of arr[0]
                                        add  r11, r11, r8
                                        add  r11, r11, r8 ;; grab addr of arr[i]
                                        ldh  r12, 0(r11) ;; load arr[i] into r12
                                        add   r10, r10, r12 ;; add arr[i] to sum

          addi r8, r8, 1 ;; increment i
          blt  r8, r9, doloop ;; evaluate loop condition
endloop:
          movia r11, sum
          sth  r12, 0(r11)  ;; write the sum into memory
```

Comment

> `gcc` is not a compiler by itself: it actually orchestrates and calls out a lot of other things
>
> 1. `cpp`: the C preprocessor, $f.c \to f.i$; dealing with `;; defines`, etc. Inspect via `gcc -E`
> 2. `gcc -s f.i`: the `cc`, the c compiler $f.i \to f.s$: parse pure c from preprocessor and spew out assembly code
> 3. `as`: the assembler, $f.s \to f.o$: to turn assembly to machine code, creating objects ready for linkage
> 4. `ld`: the linker, $f.o \to f$: link together all the objects into a single executable. Looks through `LDPATH` to look for symbols to link. [10]

[10] *There is static and dynamic linking; static meaning that everything is inside. These days most are dynamically linked which can grab symbols from shared libraries (`.so`) right before/during runtime*

Subsection 7.3

Lecture 8: Subroutines

A subroutine (or how we more commonly understand them, function calls), is a way to break up a program into smaller pieces and is a core part of structured programming.

Let's look at how we can implement the following in assembly

```c
int add3(int a, int b, int c){
        return a+b+c;
}
```

First, for `c` subroutines to work as intended they must:

1. Be callable from anywherei n the program

2. Be able to pass unique parameters across different subroutine invocations
3. Be able to return a value to the caller
4. Must be able to change control flow such that it goes back to the point where it was called

This leads to a number of questions:

1. How does the subroutine return to the caller?
2. how does it return a value?
3. How do we pass arguments?
4. What about local memory?
5. What happens to registers when we call a subroutine?

These issues are addressed through a set of implementation-specific (though largely universal) rules that all valid subroutines must follow. These are called **calling conventions**.

Definition 14 A key concept in subroutine calling is the **stack frame**. As for how stacks work/are implemented that is a google search away.
A stack has the following operations defined:

1. push: put a value on the top of the stack
2. pop: remove the top value from the stack
3. peek (distance): look at the value at a certain distance from the top of the stack

The NIOS II stores it's stack pointer, or the address to the top of the stack, in `r27`, i.e. the stack pointer `sp`. The `sp` starts at the top of the stack (highest address value) and then the stack grows downwards to lower addresses.[11] There is no way to represent an empty stack in NIOS II; `r27` will always contain a valid pointer address. [12]

[11] *Conversely the heap (commonly used for dynamic memory allocation) grows upwards towards higher addresses. The stack and heap are commonly implemented such that they grow towards each other*

[12] An element is in the stack if it's address is greater than `sp`

Definition 15 **push**

```
        ;; push r9 -> stack
        subi sp, sp, 4 ;; grow stack by a long word
        ;; (recall: grow downwards)
        stw r9, 0(sp) ;; save value of r9 to sp
```

pop

```
        ;; pop top of stack and return to r9
        ldw r9, 0(sp) ;; read top value
        addi sp, sp, 4 ;; increment sp; remove top elem
```

top

```
        // access ith elem, assuming index in r9
        add r9, r9, r9 ;; assume r9 contains nedex
        add r9, r9, r9 // r9 = 4 * i
        add r9, sp, r9 ;; sp + 4*i
        ldw r9, 0(r9) // return value of ith element into r9
```

Calling a subroutine and having it return to the caller

The stack is used to implement this behaviour. If a function will be calling another it has to save the `ra` value on the stack in the beginning and restore it from the stack prior to returning.

`r31` is commonly used as the return address pointer and is aliased as `ra`

Consider the following pseudo `C` code

```
boo_calls(){
        coo();
        doo();
        return;
}

coo(){
        doo();
        return;
}
doo(){
        return;
}
```

The NIOS II code would be as follows

Note that `ret` transfers execution to the address in `ra`

```
        .text
boo:  ;; boo will be making calls, so it first pushes the ra value on the stack
        subi sp, sp, 4
        stw ra,0(sp)  ;; push the return address onto the stack


        call coo  ;; resume execution at coo, ra = PC + 4 = boo_ret1
boo_ret1:
        call doo  ;; continue execution at doo, ra = PC + 4 = boo_ret2

boo_ret2:
        ldw  ra, 0(sp)  ;; pop return address from the stack
        addi sp, sp, 4
        ret  ;; resume execution at ra, which would be the return address from the
        ;; stack we just popped
coo:
        subi  sp, sp, 4
        stw   ra,0(sp)  ;; push the return address onto the stack
        call  doo  ;; resume execution at coo, ra = PC + 4 = coo_ret

coo_ret:
        ldw  ra, 0(sp)  ;; pop return address of boo from the stack
        addi sp, sp, 4
        ret  ;; resume execution there

doo:  ;; doo will not be making any calls, no need to save ra on the stack
        ret  ;; just return to whoever called
```

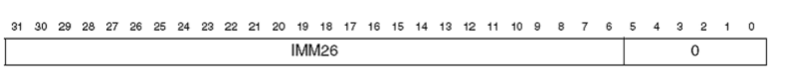


Figure 12. The call instruction accepts a label as the 2nd argument which is encoded as a 26 bit immediate. This limits the called function to be within 256 MB of the caller; the actual target is the immediate multiplied by 4 and then concatenating with the program counter, i.e. $PC_{31...28} = IMM26 * 4$

Note: we can look at executables with `objdump -d`.
Example:

```c
#include <stdio.h>
int main () {
	printf("hello world");
}
```

```

hello_world:     file format elf64-x86-64


Disassembly of section .init:

0000000000001000 <_init>:
    1000:       f3 0f 1e fa             endbr64
    1004:       48 83 ec 08             sub    $0x8,%rsp
    1008:       48 8b 05 c1 2f 00 00    mov    0x2fc1(%rip),%rax        # 3fd0 <__gmon_start__@Base>
    100f:       48 85 c0                test   %rax,%rax
    1012:       74 02                   je     1016 <_init+0x16>
    1014:       ff d0                   call   *%rax
    1016:       48 83 c4 08             add    $0x8,%rsp
    101a:       c3                      ret

Disassembly of section .plt:

0000000000001020 <printf@plt-0x10>:
    1020:       ff 35 ca 2f 00 00       push   0x2fca(%rip)        # 3ff0 <_GLOBAL_OFFSET_TABLE_+0x8>
    1026:       ff 25 cc 2f 00 00       jmp    *0x2fcc(%rip)        # 3ff8 <_GLOBAL_OFFSET_TABLE_+0x10>
    102c:       0f 1f 40 00             nopl   0x0(%rax)

0000000000001030 <printf@plt>:
    1030:       ff 25 ca 2f 00 00       jmp    *0x2fca(%rip)        # 4000 <printf@GLIBC_2.2.5>
    1036:       68 00 00 00 00          push   $0x0
    103b:       e9 e0 ff ff ff          jmp    1020 <_init+0x20>

Disassembly of section .text:

0000000000001040 <_start>:
    1040:       f3 0f 1e fa             endbr64
    1044:       31 ed                   xor    %ebp,%ebp
    1046:       49 89 d1                mov    %rdx,%r9
    1049:       5e                      pop    %rsi
    104a:       48 89 e2                mov    %rsp,%rdx
    104d:       48 83 e4 f0             and    $0xfffffffffffffff0,%rsp
    1051:       50                      push   %rax
    1052:       54                      push   %rsp
    1053:       45 31 c0                xor    %r8d,%r8d
    1056:       31 c9                   xor    %ecx,%ecx
    1058:       48 8d 3d da 00 00 00    lea    0xda(%rip),%rdi        # 1139 <main>
    105f:       ff 15 5b 2f 00 00       call   *0x2f5b(%rip)        # 3fc0 <__libc_start_main@GLIBC_2.34>
    1065:       f4                      hlt
    1066:       66 2e 0f 1f 84 00 00    cs nopw 0x0(%rax,%rax,1)
    106d:       00 00 00

                ;; and this goes on for a while loading in the .so & printf
```

And more assembly later ...

```

0000000000001139 <main>:
    1139:       55                      push   %rbp
    113a:       48 89 e5                mov    %rsp,%rbp
    113d:       48 8d 05 c0 0e 00 00    lea    0xec0(%rip),%rax        # 2004 <_IO_stdin_used+0x4>
    1144:       48 89 c7                mov    %rax,%rdi
    1147:       b8 00 00 00 00          mov    $0x0,%eax
    114c:       e8 df fe ff ff          call   1030 <printf@plt>
    1151:       b8 00 00 00 00          mov    $0x0,%eax
    1156:       5d                      pop    %rbp
    1157:       c3                      ret

Disassembly of section .fini:

0000000000001158 <_fini>:
    1158:       f3 0f 1e fa             endbr64
    115c:       48 83 ec 08             sub    $0x8,%rsp
    1160:       48 83 c4 08             add    $0x8,%rsp
    1164:       c3                      ret
```

SUBSECTION 7.4

Lecture 9

We've seen how to call and return from subroutines – but what about passing arguments and returning values? For this lecture we'll assume that only words are returned from and passed to subroutines, but other data types and structs can be used as well.

The following calling convention is the one used by gcc for the NIOS II family

- Return value is passed in `r2`[13]
- First four parameters are passed in `r4, r5, r6, r7`
- Additional parameters are pushed onto the stack *in order*
  - We push the last argument to the stack first and so forth such that the first non-register argument (the 5th one) is the first to be popped off once we enter the function.

[13] *only one return value can be given; multiple can be encoded via structures or ptrs etc*

Consider a function which takes seven integers and adds them together.

```
        .data
sum: .word 0
        .text
main:

addi sp, sp, -4
stw ra, 0(sp);
;; return address ptr (ra) is pushed onto the stack

;; fill first 4 args
movi r4, 1
movi r5, 2
movi r6, 3
movi r7, 4

;; allocated for arguments 5-7
addi sp, sp, -12 ;; 3 words * 4 byte


;; push arguments 5-7 onto the stack
;; note order; <top> 5, 6, 7
movi r2, 7
stw r2, 8(sp)

movi r2, 6
stw r2, 4(sp)

movi r3, 5
stw r2, 0(sp)

call add7

add7:
        add r2, r4, r5    # add the first two arguments and place the sum into r2
        add r2, r2, r6    # add the third argument to r2
        add r2, r2, r7    # add the fourth argument to r2
        ldw r7, 0(sp)     # read the fifth argument from the stack
        add r2, r2, r7    # add to r2
        ldw r7, 4(sp)     # read the sixth argument from the stack
        add r2, r2, r7    # add to r2
        ldw r7, 8(sp)     # read the seventh argument from the stack
        add r2, r2, r7    # add to r2 (return value in r2 by convention)
         ret
;; and then do the cleanup etc
```

Note that in reality `gcc` will actually preallocate all the memory needed by the function before the function is called. So instead of allocating 12 bytes (3x 1 word arguments) as it does on line 17, it will actually allocate 16 bytes because we need to store the return address.

The allocated stack frame for the function will be the largest of any function being called from it.

For example,

```
int main(){
        foo(1,2,3);
        boo(1,2,3,4,5,6,7,8);
}
```

Boo has the maximum number of arguments, so we'll need to allocate $8 - 4 = 4$ words on the stack for the arguments and the return address – $5 * 4 = 20$ bytes. This results in

```
;; prologue
addi sp, sp, -20
stw, ra, 16(s0)  ;; note little-endian and remaining 16 bytes for 4 argument words

;; epilogue
ldw ra, 16(sp) // pop return addr from stack
addi sp, sp, 20;
```

Subsection 7.5

Lecture 10: Recursive Subroutines

Consider this code block that computes the Ackerman function

```
int Ackerman(unsigned int x, unsigned int y)
{
        if (x==0) return y+1;
        if (y==0) return Ackerman(x-1, 1);
        return Ackerman(x-1, Ackerman(x, y-1));
}
```

This function is interesting to implement because of it's recursive nature.

Breaking it down a little more,

```
int Ackerman(unsigned int x, unsigned int y)
{
        if (x==0) return y+1;
        if (y==0) return Ackerman(x-1, 1);
        int tmp = Ackerman(x, y-1)
        return Ackerman(x-1, tmp);
}
```

The return address and the value of x must be stored on the stack, so we will need space for 2 words.

```
        .text
Ackerman:
        addi sp, sp, -8
        stw ra, 4(sp)

        bne r4, r0, Xnot0;
Xis0:
        addi r2, r5, 1
        br epilogue
Xnot0:
        bne r5, r0, Ynot0
Yis0:
        ;; pass arguments
        addi r4, r4, -1 ;;First one is x-1, i.e. r4-1
        addi r5, r0, 1 ;; second is 1
        call Ackerman
        br epilogue
Ynot0:
        stw r4, 0(sp) ;; perserve value of x on the stack
        ;; x is already at the right place for fn call
        addi r5, r5, -1 ;; decrement y
        add r5, r5, 0
        call Ackerman
epilogue:
        ldw ra, 4(s0)
        addi sp, sp, 8
        ret
```

SUBSECTION 7.6

Lecture 11: Structs and recursive structures

Consider a binary tree with a left and right child and a value. We can represent this as a struct in C.

```c
struct node{
        int value;
        struct node *left;
        struct node *right;
};
```

The memory layout of the struct is word-aligned and *exactly* like that of the struct definition. In this case value lives as addr+0, left at addr+4, and right at addr+8.

Now, consider the following recursive function to perform binary search on a BST

When this gets compiled note that in memory the struct will be word-aligned, i.e. on a 4-byte word machine the beginning of the struct will be at an address which is a multiple of the word size

```c
int findv(int d, struct node) {
        struct node * m;
        m = root;
        if (root == NULL) {
                return 0;
        }
        if (root->val == d) {
                return 1;
        }
        if (d <= root->v) {
                return findv(d, root->left);
        }
        else {
                return findv(d, root->right);
        }
        // Note that this is a tail-recursive function, i.e.
        // nothing is done after the recursive call
        // some compilers may unfurl this into a loop
}
```

How can we represent this in assembly?

```
// int d is in r4, root is in r5 (by convention)
        .text

findv:
        addi sp, sp, -4
        stw R4, 0(sp);
isrootnull:
        bne r5, r0, isdv
rootisnull:
        movi r2, 0
        br epilogue
isdv:
        ldw r2, 0(r5) ;; r2 <- root->val
        bne r2, r4, tryagain
found:
        mov r2, 1
        br epilogue
tryagain:
        ble r4, r2, goleft
goleft:
        ldw r5, 4(r5) ;; offset struct addr pointer to right member
        call findv
        br end
goright:
        ldw r5, 8(r5) ;; offset struct addr pointer to right member
        call findv
        br end
notfound:
        add r2, r0, r0
epilogue:
        ldw ra, 0(sp)
        addi sp, sp, +4
        ret
```

Subsection 7.7

Lecture 12: Devices

A computer is great and all, but for it to be useful it must communicate with the outside world via devices. A simple device interface is the Parallel Port Interface, a common implementation of which is the GPIO (General Purpose Input/Output) interface[14]

There are two registers used to communicate with the NIOSII GPIO interface which live in memory;

- `dr` (data register) at `0xFF200060`
- `dir` (data register) at `0xFF200064`

`dr` is the data word to communicate in and out of the GPIO interface, and `dir` denotes if the signal is an input or an output. Note that the `stwio` and `ldwio` variants of the `stw` and `ldw` instructions must be when interfacing with IO devices

[14] *Of which the NIOS II board has two*

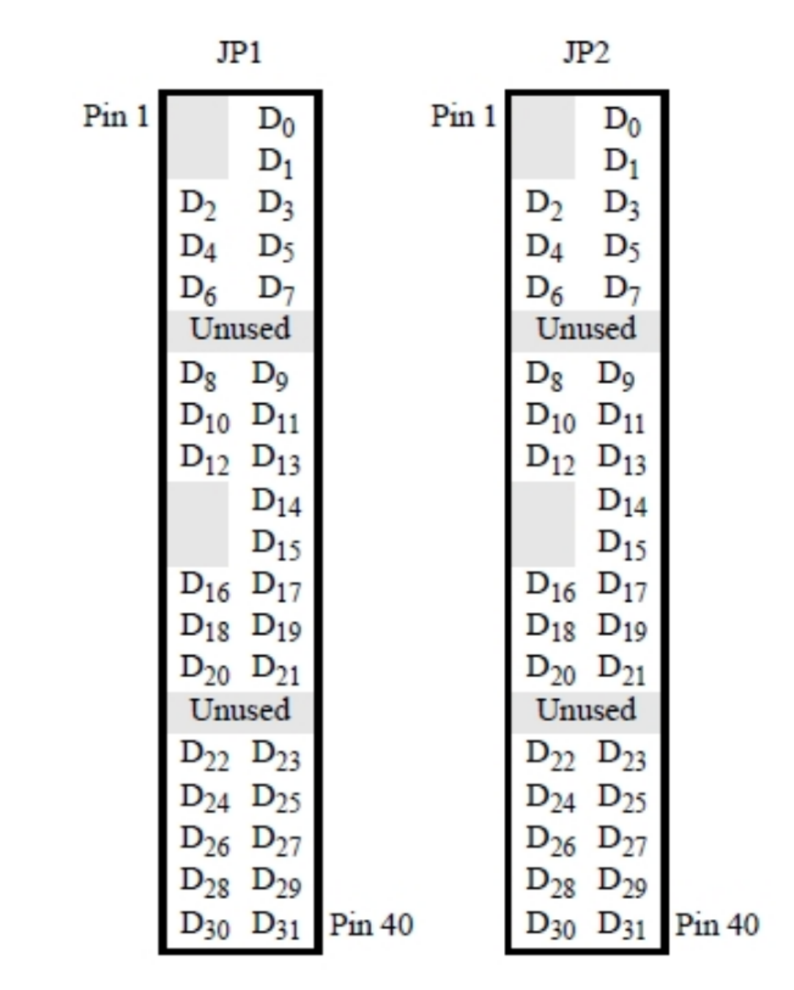


Figure 13. GPIO pinout

Two exercises we will cover include building a thermostat with the GPIO pins and building a keyboard.

Subsection 7.8

Lecture 12

PART

III

ECE355: Signal Analysis and Communication

Taught by Prof. Sunila Akbar

Section 8

Admin and Preliminary

SUBSECTION 8.1

Lecture 1

- CT and DT signals
- A ton of LTI (Linear time invariant) systems
- Processing of signals via LTI systems
- Fourier transforms
- Sampling

8.1.1 Mark Breakdown

Table 1. Mark Breakdown

Homework	20
MT1	20
MT2	20
Final	40

- Continuous enclose in (), independent is t
- Discrete: enclose in [], independent is n

In many systems we are interested in power and energy of signals over an infinite time interval; $-\infty < \{t, n\} < \infty$

Theorem 2 **Energy for Complex Signals**

$$E_{[t_1,t_2]} = \int_{t_1}^{t_2} |x(t)|^2 dt \tag{8.1}$$

$$E_{[t_1,t_2]} = \sum_{n=n_1}^{n_2} |x(t)|^2 dt \tag{8.2}$$

Average Power for Complex Signals

$$P_{avg,[t_1,t_2]} = \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} |x(t)|^2 dt \tag{8.3}$$

$$P_{avg,[t_1,t_2]} = \frac{1}{n_2 - n_1 + 1} \sum_{n=n_1}^{n_2} |x(n)|^2 \tag{8.4}$$

SECTION 9

Transformations

SUBSECTION 9.1

Lecture 2

Most of this lecture was review. When applying transforms just note to always scale, *then* shift, i.e.

1. $y(t) = x(\alpha t)$
2. $y(t) = x(\alpha t + \frac{\beta}{\alpha})$

Definition 16 **Fundamental Period**

$$x_t = x(t + mT), m \in \mathbb{Z} \tag{9.1}$$

The fundamental period, T_o is the smallest positive value of T for which this holds true

Definition 17 **Even signals**

$$x(t) = x(-t) \tag{9.2}$$

Definition 18 **Odd signals**

$$x(t) = -x(t) \tag{9.3}$$

Theorem 3 Any signal can be broken into an even and odd component

$$\begin{aligned} x(t) &= Ev\{x(t)\} = \frac{1}{2}[x(t) + x(-t)] \\ x(t) &= Od\{x(t)\} = \frac{1}{2}[x(t) + -x(-t)] \end{aligned} \tag{9.4}$$

SUBSECTION 9.2

Lecture 3

Again, most of this lecture was review from the waves portion of PHY293 from last year or some other course prior.

A complex exponential and sinusoidal system can be represented as

$$x(t) = Ce^{at} \tag{9.5}$$

Where C, a are complex numbers.

Two cases may occur.

If a imaginary and C is real we have, depending on ω, either a constant signal or a periodic sinusoidal system.

$$x(t_= e^{j\omega_0 t}) \tag{9.6}$$

- Important property: this is periodic, i.e. $Ce^{j\theta_0 t} = Ce^{j\theta(t+T)}$
- Implies that $e^{j\omega_0 T} = 1$
- Implies that for $\omega \neq 0 \rightarrow T_0 = \frac{2\overline{n}}{|\omega_0|}$

On the other hand, if a imaginary and C complex, we have a periodic signal with $T = \frac{2\overline{n}}{\omega_0}$

$$x(t) = Ce^{j\omega_0 t} = |C|e^{j\omega_0 t + \phi} = |C|\cos(\omega_o + \phi) + j|C|\sin(\omega_0 + \phi) \tag{9.7}$$

The energy of the signal is given by (8.1), or

$$E_{period} = \int_0^{T_0} |e^{j\omega_0 t}|^2 dt = \int_0^{T_0} 1 dt = T_0 \tag{9.8}$$

Recall: implication that $e^{j\omega_0 T} = 1$, therefore the quantity inside the integral evaluates to 1

$$P_{period} = \frac{E_{period}}{T_0} = 1 \tag{9.9}$$

9.2.1 General Continuous Complex Exponential Signals

The most general case of a complex exponential can be represented as a combination of the real exponential and the periodic complex exponential;

$$C = Ce^{at} \tag{9.10}$$

and

$$a = r + j\omega_0 \tag{9.11}$$

can be combined to give

Definition 19

$$Ce^{at} = |C|e^{rt}e^{j\omega_0 t+\theta} \tag{9.12}$$

Euler's relation can be used to simplify this to

$$Ce^{at} = |C|e^{rt}\left(\cos(\omega_0 t+\theta) + j\sin(\omega_0 t+\theta)\right) \tag{9.13}$$

By inspection we can see that the signal has the following properties:

1. $r = 0$: real and imaginary parts of sinusoidal
2. $r > 0$: sinusoidal signal with exponential growth
3. $r < 0$: sinusoidal signal with exponential decay

I will be skipping notes on the discrete case as it is essentially the same as the continuous case, but with the following differences

Table 2. Comparison of continuous and discrete complex exponential signals

$e^{j\omega_0 t}$	$e^{j\omega_0 n}$
Distinct signals for distinct ω_0	identical signals for distinct $\omega_0 \in \{\omega_0 \pm 2\pi i, i \in \mathbb{Z}\}$
Periodic for any ω_0	Periodic only if $\omega_0 = \frac{2\pi m}{N}$ for integers $N > 0, m$
Fundamental frequency ω_0	Fundamental frequency $\frac{\omega_0}{m}$
Fundamental period $\omega_0 = 0 \rightarrow$ undefined, otherwise $T_0 = \frac{2\pi}{\omega_0}$	Fundamental period $\omega_0 = 0 \rightarrow$ undefined, otherwise $T_0 = m\frac{2\pi}{\omega_0}$
	Since unique ω does not mean unique signal, pick $0 \le \omega_0 \le 2\pi$ or $-\pi \le \omega_0 \le \pi$

Section 10

Basic Signals

Subsection 10.1

Lecture 4: Step and Impulse Functions

One of the simplest discrete-time signals is the **unit impulse**[15] function, $\delta[n]$

[15] or unit sample

Definition 20

$$\delta[n] = \begin{cases} 0 & n \neq 0 \\ 1 & n = 0 \end{cases} \tag{10.1}$$

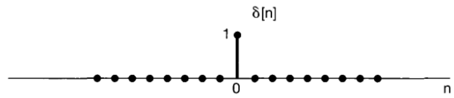


Figure 1.28 Discrete-time unit impulse (sample).

Another basic signal is the **unit step** function, $u[n]$

Definition 21

$$u[n] = \begin{cases} 0 & n < 0 \\ 1 & n \geq 0 \end{cases} \tag{10.2}$$

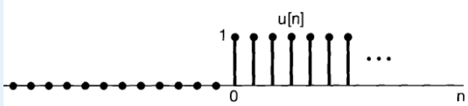


Figure 1.29 Discrete-time unit step sequence.

The unit impulse function is the first difference of the discrete time step function, i.e.

$$\delta[n] = u[n] - u[n-1] \tag{10.3}$$

And the unit step function is the running sum of the unit impulse function, i.e.

$$u[n] = \sum_{m=-\infty}^{n} \delta[m] \tag{10.4}$$

This can be rewritten with $k = n - m$ to make a more convenient expression for moving the function along $-\infty \ldots 0 \ldots \infty$

$$u[n] = \sum_{k=0}^{\infty} \delta[n-k] \tag{10.5}$$

Theorem 4 The unit impulse function $\delta[n - n_0]$ can be used to sample a function at a specific $n = n_0$ since the impulse function will take on the value 0 for all values of $n \neq n_0$

$$x[n]\delta[n-n_0] = x[n_0]\delta[n-n_0] \tag{10.6}$$

The continuous equivalents of the unit impulse and unit step functions are defined similarly

Definition 22

$$u(t) = \begin{cases} 0 & t < 0 \\ 1 & t > 0 \end{cases} \tag{10.7}$$

Likewise, the continuous unit step function is a running ~~sum~~ integral of the continuous unit impulse function

$$u(t) = \int_{-\infty}^{t} \delta(\tau) d\tau \tag{10.8}$$

A relationship analogous to the discrete case can be found for the continuous case; the continuous unit impulse function can be thought of as the first derivative of the continuous-time unit step function

Definition 23

$$\delta(t) = \frac{du(t)}{dt} \tag{10.9}$$

(10.9) is discontinuous at $t = 0$ so it is non-differentiable. We can address this by considering an approximation of (10.9) for a Δ short enough to not matter for any practical purpose

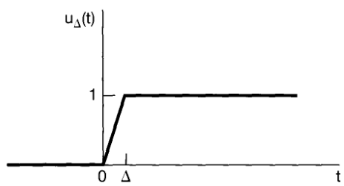


Figure 1.33 Continuous approximation to the unit step, $u_\Delta(t)$.

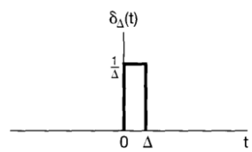


Figure 1.34 Derivative of $u_\Delta(t)$.

(10.8) can be rewritten as follows to make it more convenient to use along $\sigma \in -\infty \ldots 0 \ldots \infty$.

$$u(t) = \int_{0}^{\infty} \delta(t - \sigma) d\sigma \tag{10.10}$$

Theorem 5 And by the same argument as for the discrete case, the continuous impulse function has an important sampling property.
For any arbitrary point t_0,

$$x(t)\delta t(t - t_0) = x(t_0)\delta(t - t_0) \tag{10.11}$$

Subsection 10.2

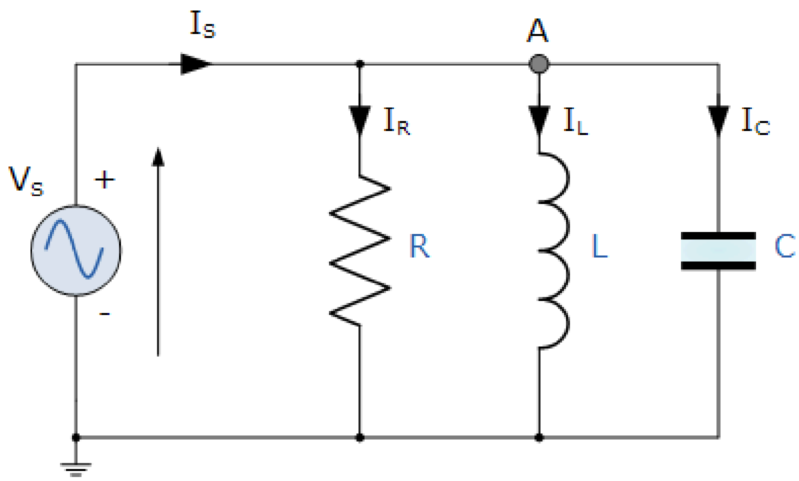
Lecture 5

Definition 24 A **system** is a process that transforms a signal, i.e.

$$x(t) \xrightarrow{\text{sys}} y(t) \tag{10.12}$$

A similar definition can be applied for the discrete case

Some examples of the relationship between signals and physical systems include circuits, i.e. RLC circuits and spring-mass systems.



Example

For example a RLC circuit can be modeled as a system that transforms a voltage signal into a current signal;

$$v(t) \xrightarrow{\text{RLC}} i(t) \tag{10.13}$$

Solving and modelling this system was covered in the other circuit classes.

Systems can be combined, i.e. making a mobile call

$$a(t) \xrightarrow{mic} y(t) \xrightarrow{antenna} z(t) \xrightarrow{tower} u(t) \xrightarrow{antenna} w(t) \xrightarrow{speaker} b(t) \tag{10.14}$$

10.2.1 Types of systems

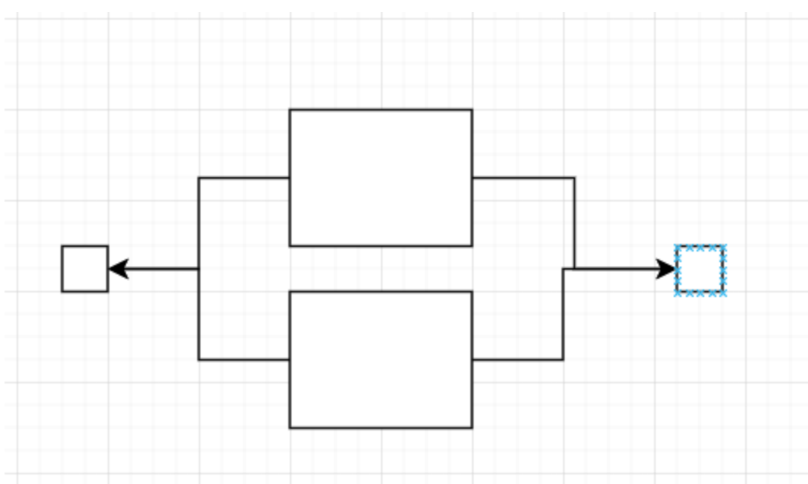


Figure 14. Parallel systems

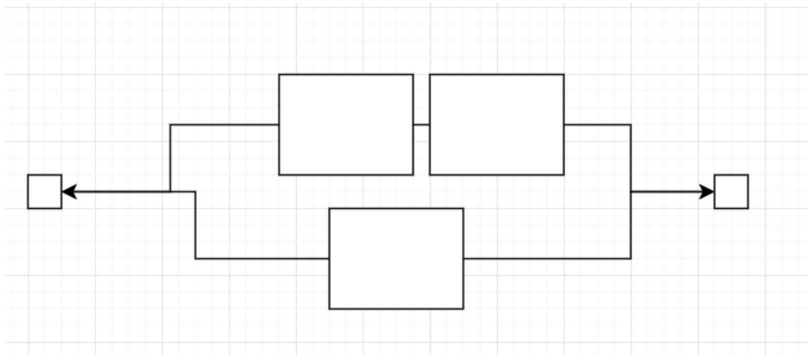


Figure 15. Series-parallel

The feedback control system will be discussed in depth in the control systems class

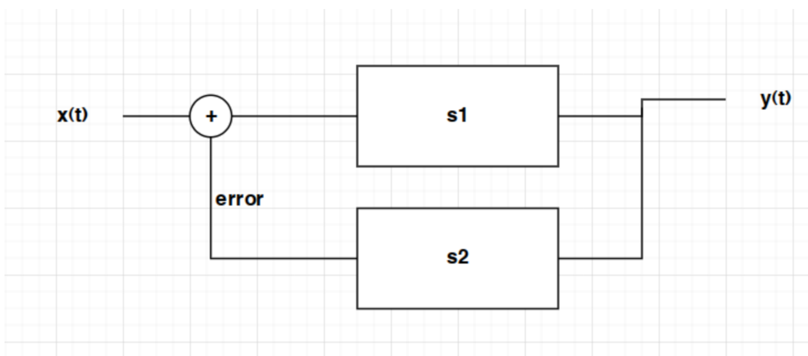


Figure 16. Feedback control system

10.2.2 System properties

Definition 25 **Memoryless** systems are systems where its output of the independent variable at a given time is dependent only on the input at the same time.
For example,

$$y[n] = 2x[n] - x^2[n] \tag{10.15}$$

is a memoryless system, but

$$y[n] = 2x[n] - x^2[n-1] \tag{10.16}$$

is not.

Other simple memoryless systems include the identity system $x(t) = y(t)$
A system with memory is the *accumulator*

$$y[n] = \sum_{k=-\infty}^{n} x[k] \tag{10.17}$$

A capacitor is an example of a continuous-time system with memory

$$v(t) = \frac{1}{C}\int_{-\infty}^{t} i(\tau)d\tau \tag{10.18}$$

There can also be systems that are dependent on future values of the input and output[16].

[16] *PID go brr*

Definition 26 A system is *invertible* if distinct inputs lead to distinct outputs[17] For example, the identity system is invertible, but the accumulator is not.

[17] *Recall: MAT185 and matrix investability*

Definition 27 A system is *casual* if its output at a given time is dependent only on the input at the current time and in the past.

Then it follows that

Lemma 2 All memoryless systems are casual

Though casual systems are useful, non-casual systems are also of great utility in modelling systems in which the independent variable is not time, or in *anticipative* models that account for the future values of the input or output, i.e a controller.

Definition 28 A system is *stable* is if the output is bounded if the input is bounded.

Definition 29 A system is *time invariant* if the behaviour and characteristics are fixed over time. For example, a RC circuit is time-invariant if the circuit R, C values are constant over time. More formally, a system is time invarient if a time shift in the input signal results in an identical time shift in the output.
If

$$x[n] \xrightarrow{\text{sys}} y[n] \tag{10.19}$$

Then, for a time invariant system,

$$y[n-n_0] \xrightarrow{\text{sys}} x[n-n_0] \tag{10.20}$$

Definition 30 A system is *linear* if it possess the property of superposition, i.e. it possess the additive property and the scaling, or homogeneity property.
More formally, a linear system is one such that

$$ax_1(t) + bx_2(t) \xrightarrow{\text{sys}} ay_1(t) + by_2(t) \tag{10.21}$$

Where y_1 is the output of a system with input x_1 and y_2 is the output of a system with input x_2.

SECTION 11

LTI systems

SUBSECTION 11.1

Lecture 6: Discrete LTI systems

Many physical systems can be modelled as linear time-invariant (LTI) systems. This is useful because there exists a large body of theory that can be applied to LTI systems, in part due to LTI systems possessing the superposition property.

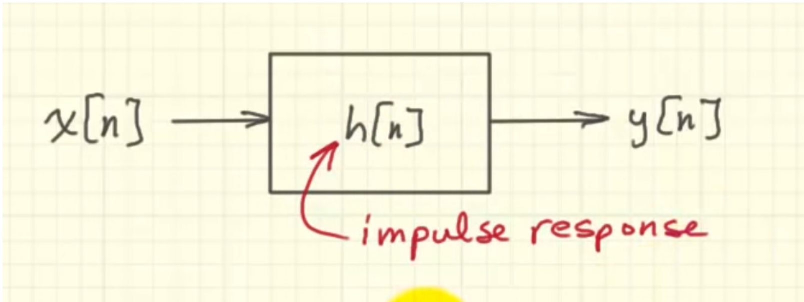
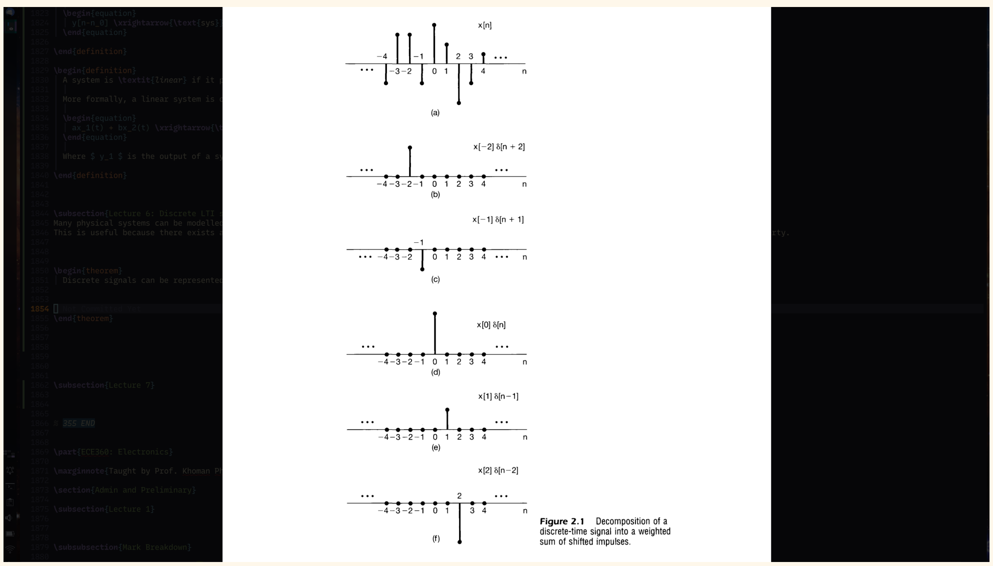


Figure 17. In this lecture the idea of LTI systems being a sum of $h[n]$ impulse responses will be

Theorem 6 Discrete signals can be represented as a summation of impulses.



More generally, a discrete signal can be rewritten as a sum of shifted unit impulses with weights $x[k]$

$$x[n] = \sum_{n=-\infty}^{\infty} x[k]\delta[n-k] \tag{11.1}$$

Example As an example we can rewrite the unit step function as

$$u[n] = \sum_{n=0}^{\infty} \delta[n-k] \tag{11.2}$$

The discrete unit impulse also possesses a *sifting* property; since $\delta[n-k]$ is nonzero only when $k = n$, this summation will sift through the values of $x[k]$ and preserves only the values corresponding to $k = n$

A useful result of the properties of LTI systems (i.e. the superposition property) is that the output of a system is the convolution of the input and the impulse response of the system, i.e.

Definition 31 **Convolution** sum

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k] \tag{11.3}$$

Theorem 7 An LTI system can be modelled as a **convolution** sum, or a sum of scaled and shifted impulse responses. So, if we know the response of the linear system to the set of shifted unit impulses – we can determine the response of the system to any input signal!

Symbolically convolution can be written as

$$y[n] = x[n] * h[n] \tag{11.4}$$

Subsection 11.2

Lecture 7: Continuous LTI systems

The concept of a LTI system responding to unit impulses and taking advantage of the sifting property may be extended to the continuous systems by idealizing the pulse as one so short that it's duration is inconsequential for any real system.

Definition 32 **Sifting property** of continuous-time impulse

$$x(t) = \int_{-\infty}^{\infty} x(\tau)\delta(t-\tau)d\tau \tag{11.5}$$

Can interpret $x(\tau)\delta(t-\tau)$ equals $x(t)\delta(t-\tau)$, i.e. that it is a *scaled impulse*

$$x(t) = \int_{-\infty}^{\infty} x(\tau)\delta(t-\tau)d\tau = x(\tau)\int_{-\infty}^{\infty} \delta(t-\tau)d\tau \tag{11.6}$$

If we let $h_\tau(t)$ denote the response at time t to an unit impulse $\delta(t-\tau)$ at time τ, then

$$y(t) = \lim_{\Delta\to 0} \sum_{k=-\infty}^{\infty} x(k\Delta)\hat{h}_{k\Delta}(t)\Delta \tag{11.7}$$

As $\Delta \to 0$ the summation becomes an integral, i.e.

$$y(t) = \int_{-\infty}^{\infty} x(\tau)h_\tau(t)d\tau \tag{11.8}$$

For notational convenience the τ subscript can be dropped and generalizing to all t

$$y(t) = \int_{-\infty}^{\infty} x(\tau)h(t-\tau)d\tau \tag{11.9}$$

Symbolically the convolution is represented $y(t) = x(t) \cdot h(t)$

Subsection 11.3

Lecture 8

PART

IV

ECE360: Electronics

Taught by Prof. Khoman Phang

Section 12

Admin and Preliminary

Subsection 12.1

Lecture 1

12.1.1 Mark Breakdown

Table 3. Mark Breakdown

Test 1	15
Test 2	20
Homework	10
Labs	12
Final	43

12.1.2 Diodes

Diodes are an electronic valve which causes current to only flow in one direction. An ideal diode is an open circuit in the closed direction and a closed circuit in the other, so the current is always in the direction of the arrow (+'ve @ arrow base, -'ve at arrow point)[18].

[18] *recall: passive sign convention*

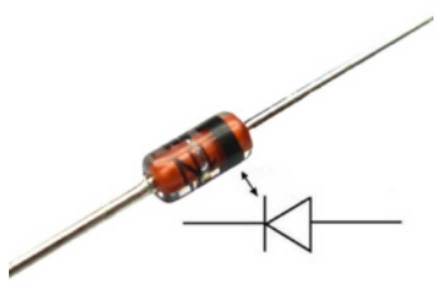
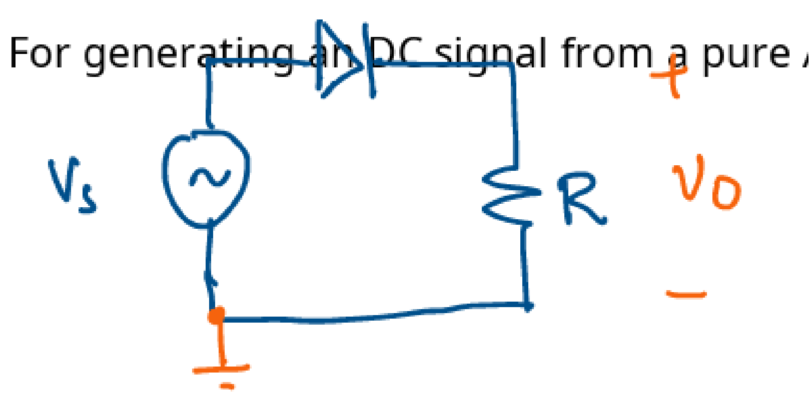
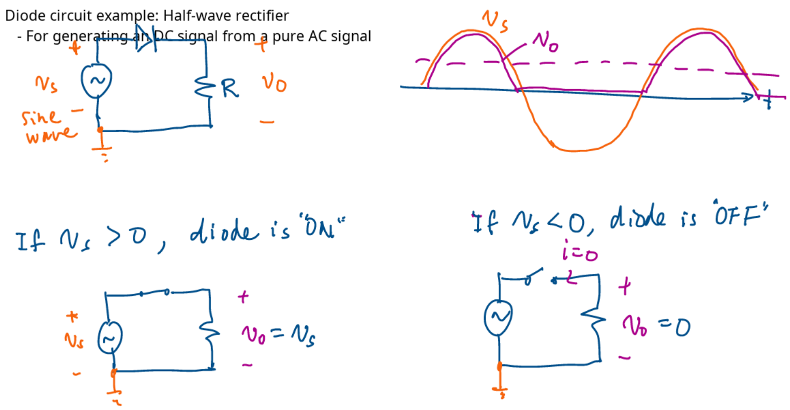


Figure 18. A diode and its symbol

An example of a diode circuit is the half-wave rectifier which turns an AC signal to a DC signal

Can take oscilloscope over resistor to see that a pure DC signal has been generated





Section 13
Diodes

Subsection 13.1
Lecture 2

More formally, off/on for diodes should be referred to as:

- Off ↔ reverse bias
- On ↔ forwards bias

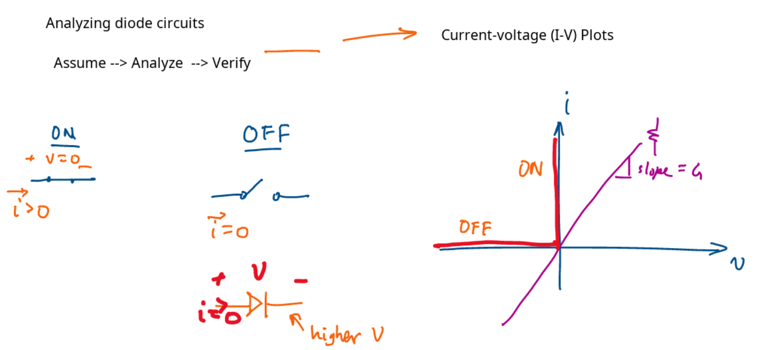


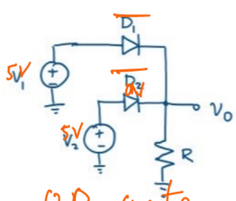
Figure 19. General steps for analyzing non-linear circuits. Note plotting out expected response

An example of how this is used in circuit design is to manage two power sources. Consider an Arduino that could be powered by an AC adapter or by a computer's USB port. This circuit would choose the higher voltage source and prevent backflow into the other power source due to any potential power differentials. It is also effectively an OR gate

Analysis Examples

'0' '1'

Example 1:
Find output voltage Vo assuming input voltages V_1 and V_2 are either 0V or 5V.
What is the function of this circuit?

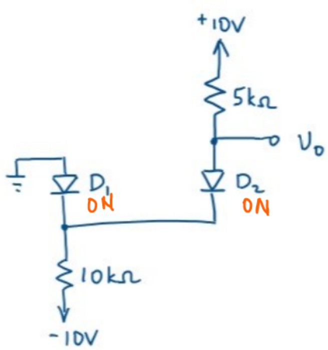


OR gate

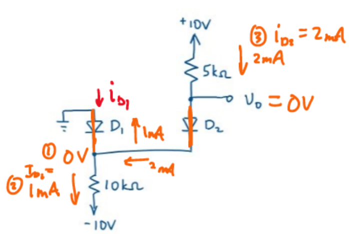
V_1	D_1	V_2	D_2	V_o
5V	ON	5V	ON	5V
5	ON	0	OFF	5V
0	OFF	5	ON	5V
0	OFF	0	OFF	0V

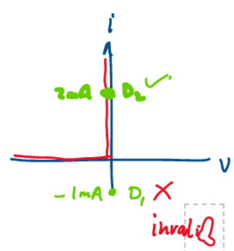
Example 2:
Find output voltage Vo

Assume --> Analyze --> Verify

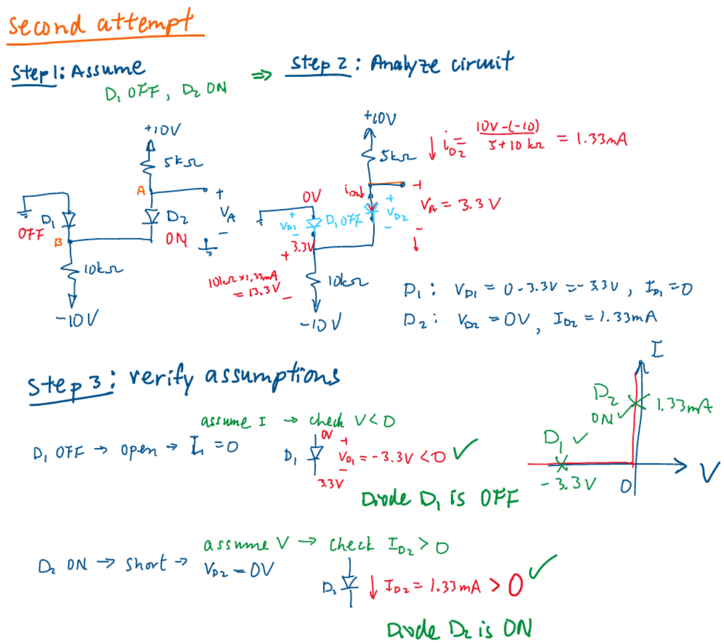


Assume D_1 ON
D_2 ON





In this example the initial assumption was incorrect.
Let's try another analysis with D_1 off and D_2 on:



If we were to do this brute force we'd have to consider 4 cases, so it's important to build up some sort of intuition for the circuit.

Subsection 13.2
Lecture 3

Today we're going to look at the characteristics of real diodes.

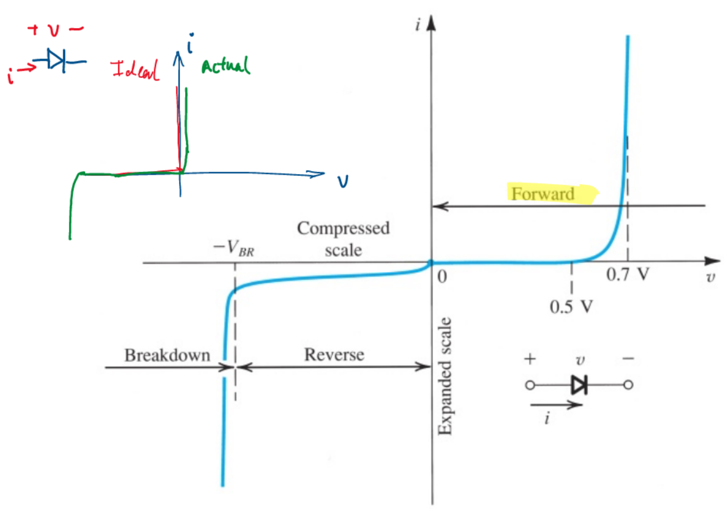


Figure 4.8 The silicon diode i–v relationship with some scales expanded and others compressed in order to reveal details.

Real diodes have a little bit of leakage current and also encounter a breakdown point where they're no longer able to block the current.

Theorem 8 **Forward Bias**

$$i = I_s(e^{\frac{V}{V_T}-1}) \tag{13.1}$$

Where:

$$V_T = \frac{kT}{q} \quad [V] \tag{13.2}$$

k is Boltzmann's constant, T is temperature in Kelvins, q is the charge of an electron.
I_s is the scale current which is usually $\approx 1pA$, which doesn't change much until the breakdown point.

Most of the time we can assume that the circuit is at room temperature and that $v_T = 25mV$. Note that this value explodes when $V > V_T$ which is the breakdown point. When encountering a reverse bias $V_s < 0$, the -1 term comes in and causes $i \approx I_s$
The scale current is just a general constant which varies in range from $10^{-9} to 10^{-15} A$ and scales with temperature, doubling with every approximately $5^o C$ increase in temperature.
Note: the ideal diode equation can be rearranged to find an expression for voltages

$$V = V_T \ln{(\frac{i}{I_s})} = \ln{(10)} V_T \log_{10}{(\frac{i}{I_s})} \tag{13.3}$$

These expressions turns out to be quite reliable for reasonable diodes to reasonable voltages.

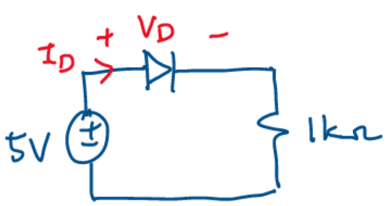
Using the ideal diode equation we can find the relationship between voltages and currents as they pass through the diode.

$$\frac{i_2}{i_1} = \frac{I_s e^{\frac{V_2}{V_T}}}{I_s e^{\frac{V_1}{V_T}}} = e^{\frac{V_2 - V_1}{V_T}} \tag{13.4}$$

$$V_2 - V_1 = V_T \ln{(\frac{i_2}{i_1})} \xrightarrow{\text{room temperature}} 60mV \log_{10} \frac{i_2}{i_1} \tag{13.5}$$

Example:

Calculate the diode voltage and current in the circuit below.
Assume that the diode voltage is 0.7V at 1mA and V_T = 25mV.



Example Recall (13.1). Plugging in the given values gives us the scale current.

$$1mA = I_s e^{\frac{0.7V}{25mV}}, I_s = 6.9 \cdot 10^{-16} A \Rightarrow I_o = I_s e^{v_o / v_T} \tag{13.6}$$

Ohm's law can then be applied at the resistor

$$V_r = IR = I_o R = 5V - V_D \Rightarrow 5 - V_D = I_o R \tag{13.7}$$

V_D is the voltage across the diode

So we have two equations and two unknowns (since we know $v_T = 25mV$ but v_o was used at first just to find I_s) Solving for the unknowns gives us:

- $V_o = 0.736V$

- $I_D = 4.264mA$

Section 14

Lecture 4 & 5: Forward conducting diodes

The exponential model accurately describes the diode outside of the breakdown region, though it's nonlinear behaviour makes it difficult to use.

V_{DD} is DC voltage, v_D is small signal voltage, V_D is the diode voltage

For $V_{DD} > 0.5V$

$$I_D = I_S e^{V_D/V_T} \tag{14.1}$$

Where

- I_S is the diode parameter
- V_T is the thermal voltage

Another equation may be produced via Kirchhoff's law

$$I_D = \frac{V_{DD} - v_d}{R} \tag{14.2}$$

The unknown quantities I_D and v_d may be solved for via graphical analysis or iteration.

Example This simple circuit is used to demonstrate the exponential model of the diode.

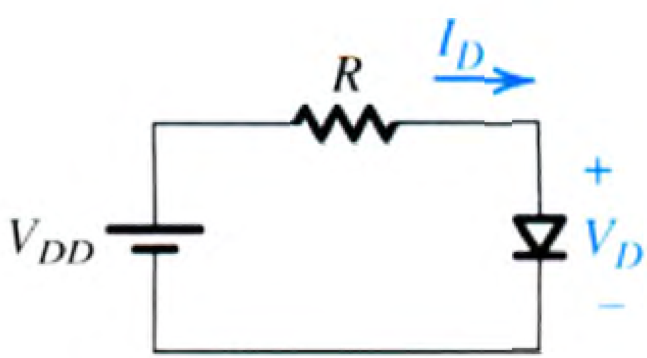
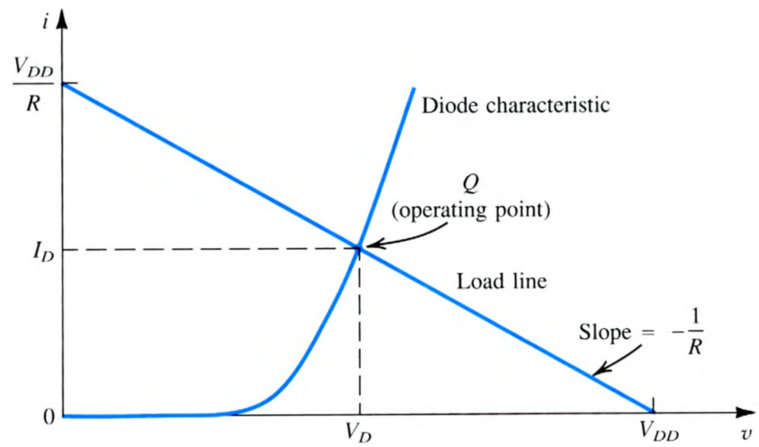


Figure 20. Simple example circuit with diode

Plots of the diode characteristics and Kirchhoff's relation are plotted, the intersection of which gives the solution.



An iterative procedure may also be applied to solve for the unknowns, the procedure for which will be illustrated through an example

Example Find I_D, V_D for the circuit in the previous example (Fig. 20). $V_{DD} = 5V, R = 1k\Omega$, and at $V_D 0.7V$, $I_D = 1mA$

1. Assume $V_D = 0.7V$, then use (14.2) to find I_D.

$$I_D = \frac{5V - 0.7V}{1k\Omega} = 4.3mA \tag{14.3}$$

2. Use the diode equation (14.1) to get a better estimate for V_D.

$$V_2 - V_1 = 2.3V_T \log \frac{I_2}{I_1} \Rightarrow V_2 = V_1 + 0.06 \log \frac{I_2}{I_1} \tag{14.4}$$

substituting $V_1 = 0.7V, I_1 = 1mA, I_2 = 4.3mA$,

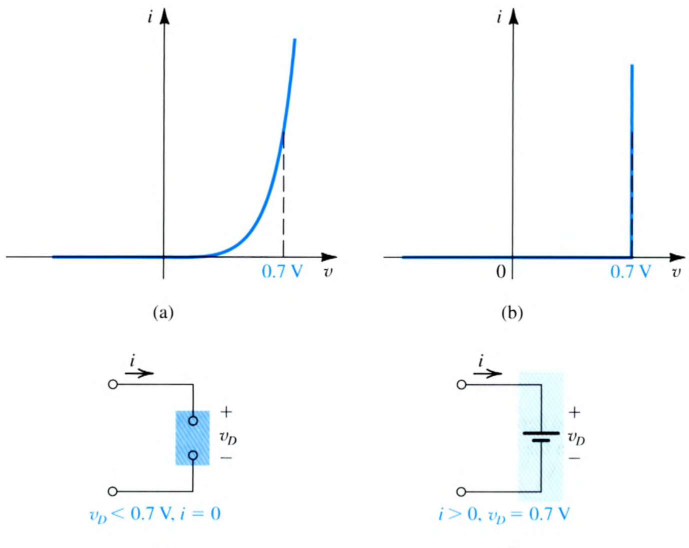
$$V_2 = 0.738V \Rightarrow I_D = 4.3mA, V_D = 0.738V \tag{14.5}$$

- This states that for a decade[19] change in current the diode voltage drop changes by $2.3(V_T \approx 60mV)$ which is negligibly small for $v < 0.5V$. The voltage at which the this behaviour becomes significant is called the **cut-in voltage**

3. Repeat steps 1 and 2 with the new values until the values more or less become stable

[19] *Factor of 10*

This iterative model is powerful and yields accurate results, but can be computationally expensive especially when calculating by hand. To address this we employ other models such as the *constant-voltage-drop* model which approximates the exponential characteristics via a piecewise linear model. The reason why this is possible is because forward conducting diodes exhibit a voltage drop that varies in a relatively narrow range.



Using the constant voltage drop model in our analysis looks the same as before, but with V_D directly taking on the value of $0.7V$ (as per the prior example) instead of being solved for with the diode equation.

In applications that involve voltages greater than the voltage drop (i.e. usually $\approx 0.6 - 0.8V$) we can neglect the diode voltage drop altogether while calculating the diode current.

$$\begin{aligned} V_D &= 0V \\ I_D &= \frac{5-0}{1} = 5mA \end{aligned} \tag{14.6}$$

This is generally good enough for a first estimate, though the previous model isn't that much more work and gives more accurate results. The primary use of this model is to determine which diodes are on or off in a multi-diode circuit

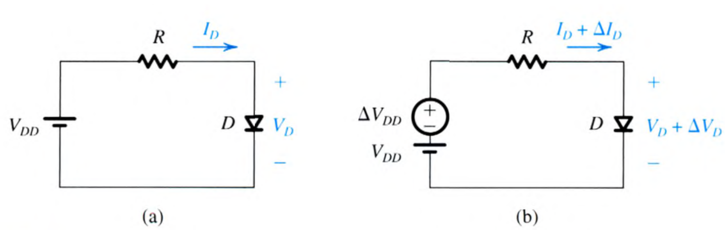
Section 15

Small-Signal Model

The small signal method is an alternative model used to describe the nonlinear diode's characteristics with greater accuracy than piecewise linear models.

Similar methods will be applied to transistors in later chapters

Consider a small ΔV_{DD} applied to the diode, which would cause a small $\Delta I_D, \Delta V_D$. We want to find a quick way of determining the values of these incremental changes.



Skipping a bunch of math[20] the results are as follows: [21]

[20] It is 11:17pm and I have two more lectures to catch up to today

[21] Small signal analysis can be performed separately from the dc bias analysis because of the linearization of diode characteristics in the small-signal ap-

Definition 33 **Small signal approximation**

$$i_D(t) \approx I_D(1 + \frac{v_d}{V_T}) \tag{15.1}$$

This is valid for when variations in diode voltage $|v_d| \lessapprox 5mV$.

$$r_d = \frac{V_T}{I_D} \tag{15.2}$$

From this we can define the small signal resistance as the resistance relating i_d to v_d

The steps for calculating the small signal model are as follows:

1. Perform a dc analysis using the exponential, constant-voltage-drop, or piecewise-linear model.
2. Linearise the circuit. For a forward-based diode, find r_d by substitution I_D into (15.2). The small-signal equivalent circuit is found by eliminated all independent dc sources[22] and replacing the diode with its small-signal resistance r_d
3. Solve the linearised circuit. In particular we would want to find $\Delta I_D, \Delta V_D$ and check to see if it is consistent with our approximation, i.e. that $\Delta V_D \lessapprox 5mV$

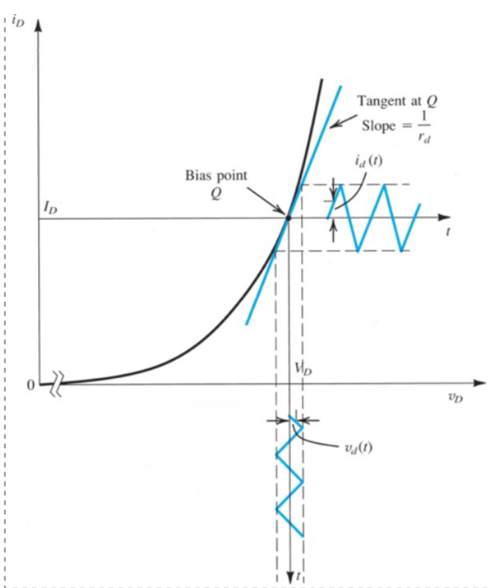
[22] since we already accounted for them in step 1

The reason why we linearise these non-linear systems, we, as engineers, try to linearise them because it is convenient to be able to use superposition, phasors, Fourier series, Laplace transforms, and so forth.

Section 16

Lecture 6: Small signal model, cont'd

V_D can be thought of an input to a transfer function that is the diode, with I_D being the output. If the input signal is more or less a triangular wave the output will be as well.



Since we are applying a linear approximation, superposition may apply;

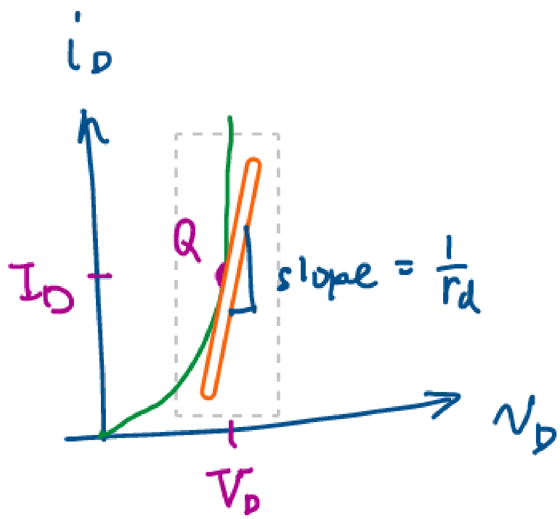
$$f(a+b+c\dots) = f(a) + f(b) + f(c) + \dots \tag{16.1}$$

Here we can think of the function as

$$f(V_o + v_d) = f(V_o) + f(v_d) \tag{16.2}$$

Small signal analysis works because, by superposition, we can zero out the other sources (V_o, etc) and inspect only the effect of the small signal voltage on the system.

16.0.1 Deriving small-signal resistance



Resistance can be found as the slope of the $I-V$ relationship. The following is a proof of (15.2).

PROOF

$$i_D = I_s e^{\frac{V_D}{V_T}} \tag{16.3}$$

$$\text{slope} = \frac{di_D}{dv_o} = I_S(\frac{1}{v_T})e^{\frac{V_D}{V_T}} \tag{16.4}$$

Then, substitute values at the operational point Q , i.e. I_D, V_{DD}

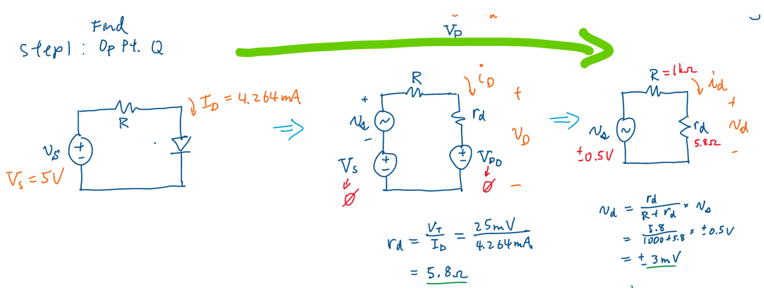
$$\text{slope} = I_S e^{V_D/V_T}(\frac{1}{V_T}) \implies \frac{I_D}{V_T} = \frac{1}{r_d} \tag{16.5}$$

And therefore

$$r_d = \frac{V_T}{I_D} \tag{16.6}$$

□

1. Calculate DC bias point Q
2. Derive small-signal circuit
3. Analyze small-signal circuit
4. If required, recombine to arrive at final result



The textbook skips the middle step where we actually apply the small signal approximation and all the values have not been offset by the bias point yet. Note that voltages/values in the 3rd step (small signal approx) are all relative to the bias point Q .

Applying this to our circuit we find

$$r_d = \frac{V_T}{I_D} = \frac{25mV}{4.264mA} = 5.8\Omega \tag{16.7}$$

This was not as accurate as the exponential model but it is more than close enough 99% of the time. But also the $0.7V$ constant voltage drop model is usually sufficient as well.

PROOF The small-signal approximation is valid for voltage variations up to $\pm 5mV$

$$\begin{aligned} i_D &= I_S \exp(V_D/V_T) \\ &= I_S \exp \frac{V_D D + v_d}{V_T} \\ &= I_D \times e^{v_d/v_T} \end{aligned} \tag{16.8}$$

e^x can be expanded to a power series

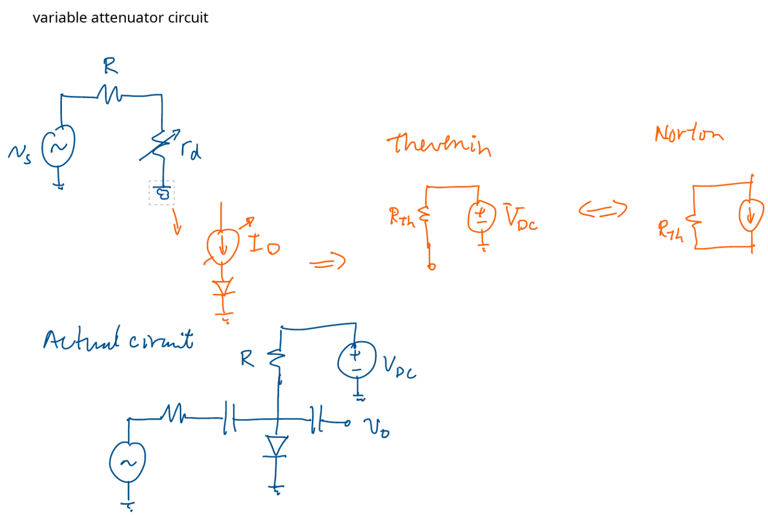
$$\begin{aligned} e^x &= 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots \\ &\approx 1 + x \qquad \text{if } \frac{x^2}{3!} \ll x \to x \ll 2 \end{aligned} \tag{16.9}$$

Therefore $\frac{v_d}{V_T} \ll 2$, so $v_d \ll 2V_T = 50mV$
So make

$$|v_d| < 5mV \tag{16.10}$$

□

In the lab we'll be using a variable attenuator circuit, which involves a voltage source Can make a current source with a voltage source and a large resistor.

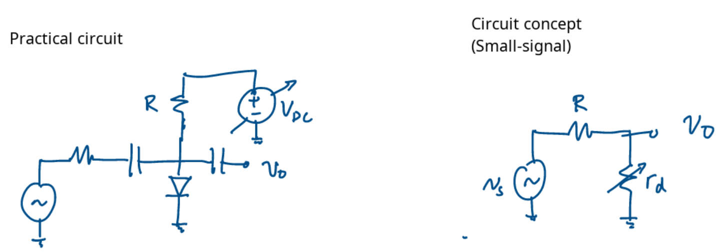


This will be covered more next lecture. TLDR: can use small signals and then build up a circuit to get the intended Q using biases.

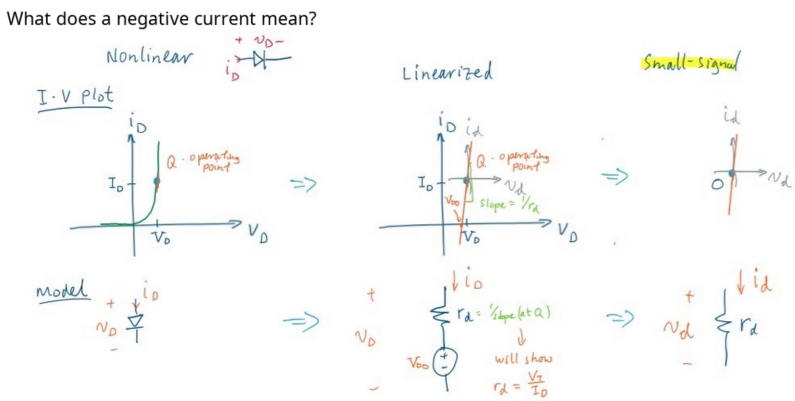
Section 17

Lecture 7

Let's take a deeper look into the variable attenuator circuit discussed last class.



- The capacitors in this circuit can be treated like short circuits because they are large (which is why we don't see them in the small-signal circuit).
- The constant voltage offset source gets zeroed out in the small-signal circuit because it is a constant, thereby becoming a ground.
- Current sources become open circuits
- Negative currents

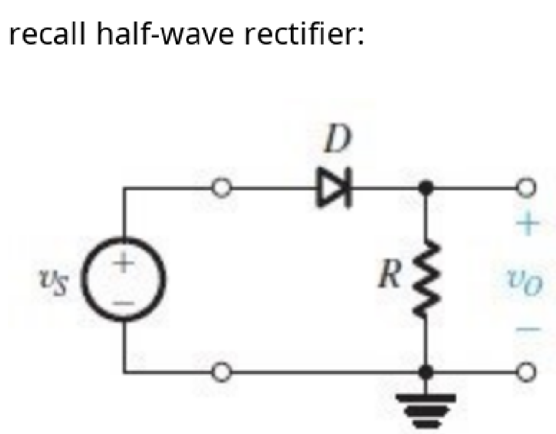


Section 18

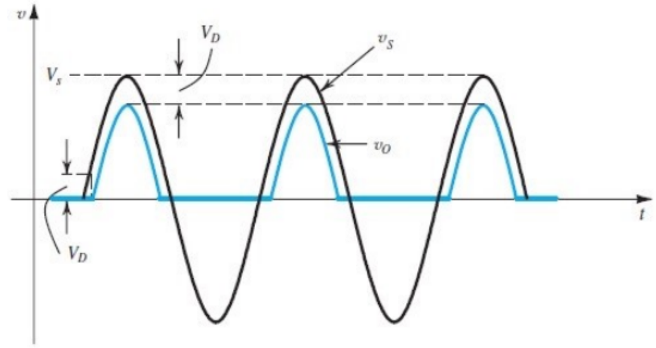
Rectifiers

Most of the homework, etc we will be practicing working from the practical circuit to the small-signal design, whereas as a designed we will usually be going the other way around.

We have previously talked about the half-wave rectifier circuit, which used a single diode to rectify a waveform to only the positive values



This isn't very efficient because it only uses half of the waveform, so it is desirable to build a *full wave* rectifier



What we want with a full-wave rectifier is to automatically interchange the wires whenever the input signal goes from negative to positive and vice-versa as to provide the load a wholly positive signal.

A full wave rectifier can be built as follows

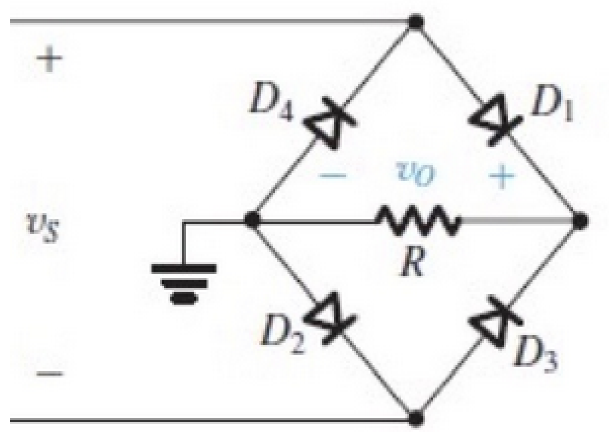


Figure 21. When the signal is positive it will flow from diode 1 to the load and then to diode 2. When it is the negative polarity it will go through diode 3 to the load and then return through diode 4. Note how the direction remains the same, and how we selectively connect the ground to the top or bottom terminal depending on the input polarity.

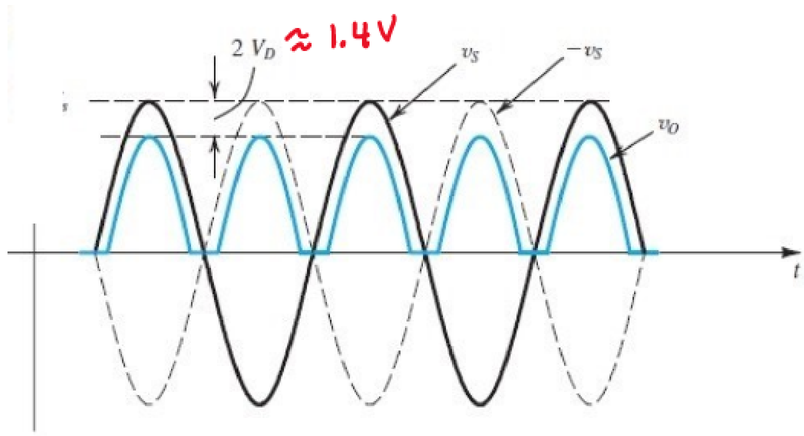


Figure 22. Comparing the two there is a little more loss but at least we get the full wave now

A **peak rectifier** uses a capacitor as an energy storage device to hold the voltage at the peak value.

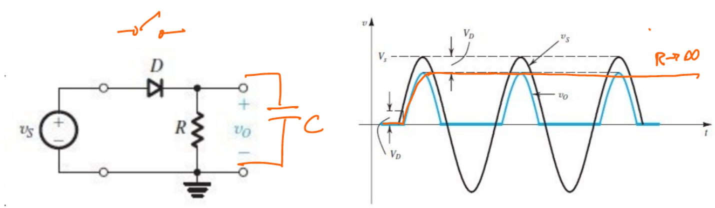
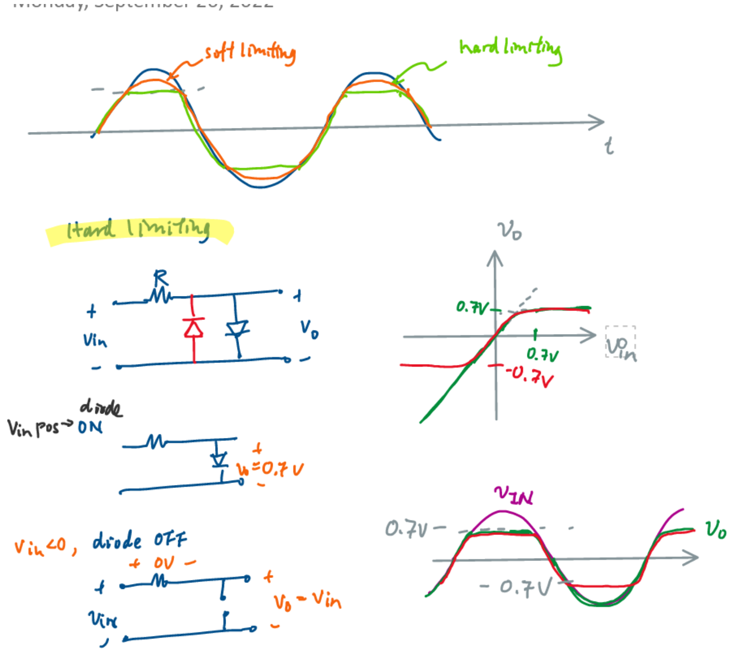


Figure 23. Capacitor size and frequency must be tuned as to avoid voltage drop (recall RC circuits. Note that the time constant $\propto RC$, so a bigger resistor (slower rate of discharge) or bigger capacitor (bigger energy storage) can reduce voltage drop)

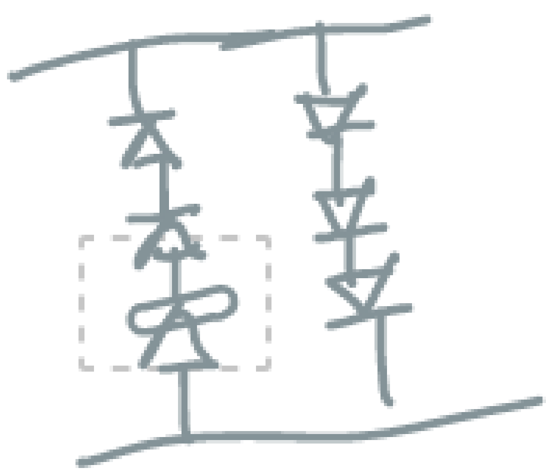
SUBSECTION 18.1

Lecture 8

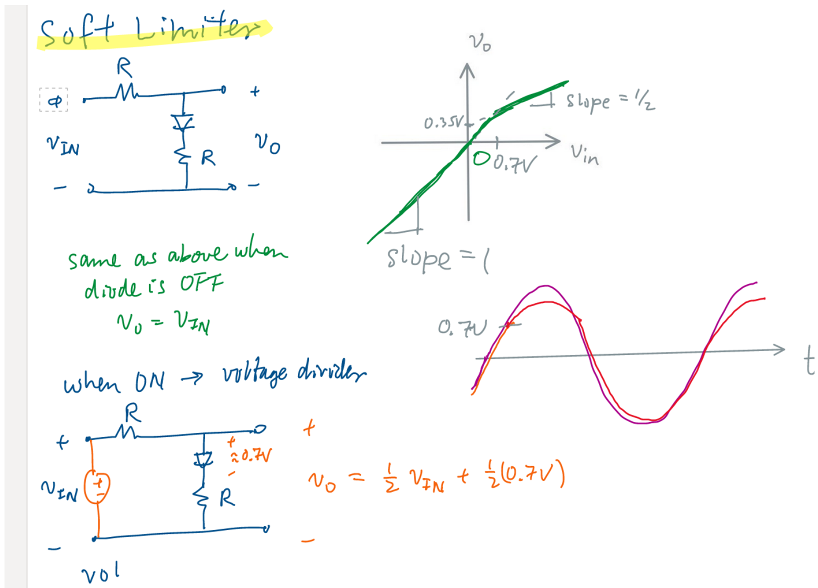
Hard limiting circuits include a pair of diodes which cause only a maximum of $0.7V$ signal to pass through.



Increasing the hard limiting voltage can be done by putting a bunch of diodes in series, i.e.



The hard, squared-off signal response (and multiple of 0.7V) may not be desirable, so we can use a soft limiter instead.



This will act like nothing is happening when the diode is off, i.e. $v_o = v_{in}$. When the diode is on the soft limiting circuit acts like a voltage divider. For this circuit there are two resistors with the same R so the output voltage from just the current divider is

$$v_o = \frac{R}{R+R} v_{in} = \frac{1}{2} v_{in} \tag{18.1}$$

We will also have to account for the voltage of the diode, so

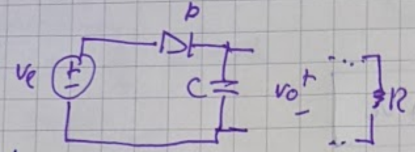
$$v_o = \frac{1}{2} v_{in} + \frac{1}{2}(v_d = 0.7V) \tag{18.2}$$

$$v_o = \frac{1}{2} v_{in} + \frac{1}{2} 0.7V \tag{18.3}$$

Subsection 18.2
Lecture 9

I missed this lecture, so here are my handwritten catchup notes

Clamped capacitor & ~~Peak Rectifier.~~ $\boxed{\text{PEAK RECTIFIER}}$

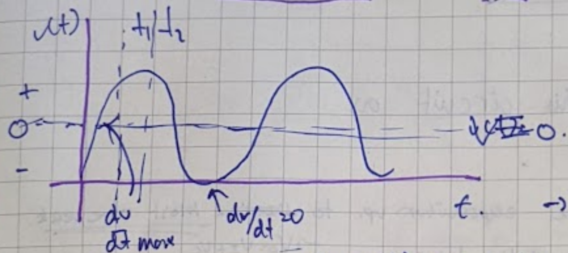


Assume time constant $RC >> T$.

$i_L = v_o/R$ (load current).

$i_D = i_C + i_L$ (kirchoff's)

$= C\frac{dv}{dt} + i_L$ ☆



$\frac{dv}{dt}$ @ max

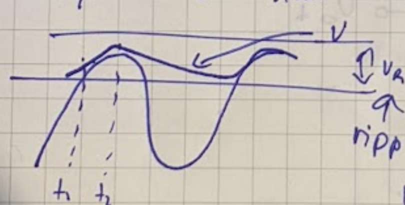
→ observe dv/dt trends w/ value of $v(t)$.

@ $v(t) > 0$, the diode will allow current through and therefore charge the capacitor.

@ $v(t) < 0$, the diode will not allow current through → capacitor will discharge.

Note relationship between i_C & dv/dt. the contribution due to the capacitor will be @ max when v is close to zero since that is where dv/dt is @ max, and vice versa.

[From figure] will start conducting @ t_p & stop @ t_2.



☆ t_2, t_1 can be found with (☆) ($i_D = C\,dv/dt + i_L$).

ripple voltage.

$$V_R = \frac{V_P}{fCR} = \frac{I_L}{fC}.$$

$$\Delta t = t_2 - t_1 = V_p\cos(\omega\Delta t) \underset{\text{small } \theta}{\approx} \frac{T}{2\pi}\sqrt{\frac{2V_r}{V_p}}.$$

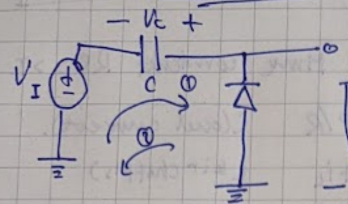
Another interpretation:
charge supplied by ~~diode~~ capacitor must equal the charge gained by the diode.

$$Q_{supply} = i_C\,\Delta t \simeq i_{D_{avg}} - I_L\frac{T}{2\pi}\sqrt{2V_r/V_p} = Q_{lost} = CV_R = I_L T$$

☆ energy charged into capacitor must equal that lost during steady state.

Pick diode w/ $i_{D_{max}} = I_L(1 + 2\pi\sqrt{2V_p/V_r})$

CLAMPING CAPACITOR



$$\boxed{V_o = V_I + V_c}$$

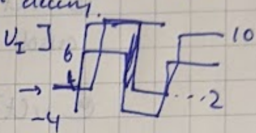
Can visualize response to this circuit as the diode preventing .

① current in ① direction: charge capacitor up to ~~...~~ most −ve peak.

② current in ②: w/ $R_L = \infty$, straight line but exp decay. $\to V_o = V_I + V_c$

→ output voltage @ $V_C = [V_I]$

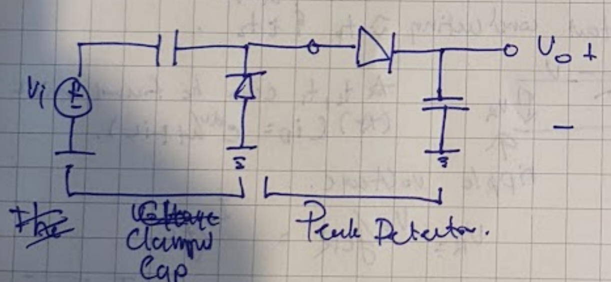
ie w/ $V_I = -4V \leftrightarrow 6V$ →



~~A~~ same waveform,

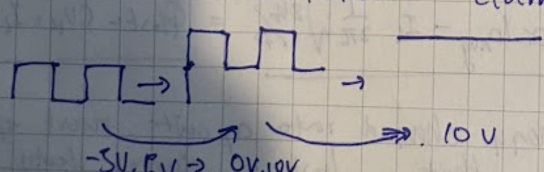
→ Feeding this clamped capacitor circuit to a clamping circuit provides a nice DC signal.

Voltage Doubler

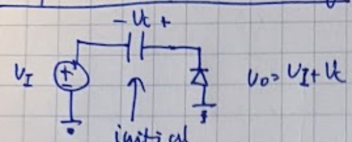


Voltage doubler $V_0 = 2V_i$

Intuition: Clamped cap doubles input signal, Peak Detector clamps @ 2x!

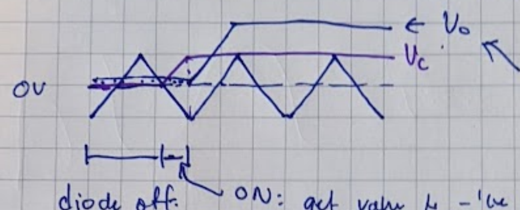


Example Clamper Analysis

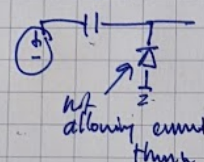


Input: 5V triangular.

$V_o = V_I + V_C = V_I + C\frac{dv}{dt}$.

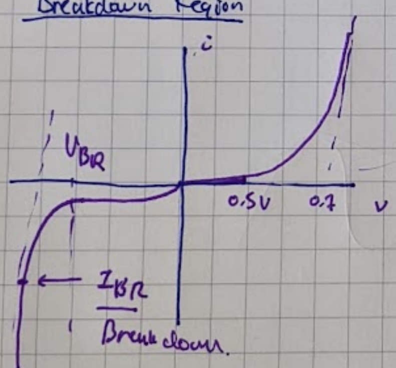


$V_o = V_i + V_c$



Why does it only charge when circuit ↻, not ↺ ?

Breakdown Region

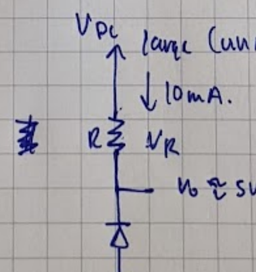


"Zener Voltage"

Typical $V_{BR} \gtrsim 200V = V_Z$

Operation $\approx V_Z$ is not destructive for small I

- can be used to generate voltage.



$R = \frac{V}{I} = \frac{7}{10mA} = 700\Omega$

Desired regulated voltage may be created by exploiting the breakdown current

$(V_o = V_Z) + V_R = V_{DC}$.

∴ Pick V_R s.t. $V_Z = V_{DC} - V_R$

so pick $\boxed{R = V_R/I}$

ECE358: Foundations of Computing

PART

V

Taught by Prof. Shurui Zhou

Section 19

Admin and Preliminary

Subsection 19.1

Lecture 1

Topics covered will include:

- Graphs, trees
- Bunch of sorts
- Fancy search trees; red-black, splay, etc
- DP, Greedy
- Min span tree, single source shortest paths
- Maximum flow
- NP Completeness, theory of computation
- Blockchains??
- Θ

Solutions will be posted on the window of SF2001. Walk there and take a picture.

19.1.1 Mark Breakdown

Table 4. Mark Breakdown

Homework x 5	25
Midterm (Open book)	35
Final (Open book)	40

Section 20

Complexities

Subsection 20.1

Lecture 2

This lecture we talked about big O notation. For notes on this refer to my tutorial notes for ESC180, ESC190: https://github.com/ihasdapie/teaching/

Definition 34 Big O notation (upper bound)
$g(n)$ is an asymptotic upper bound for $f(n)$ if:

$$O(g(n)) = \{f(n) : \exists \quad c, n_0 \quad s.t. \quad 0 \leq f(n) \leq c \cdot g(n), \forall n \geq n_o\} \tag{20.1}$$

PROOF **What is the big-O of $n!$?**

$$n! \leq n \cdot n \cdot n \cdot n \ldots n = n^n \Rightarrow n! \in O(n^n) \tag{20.2}$$

□

Definition 35 Big Ω notation (lower bound)
$h(n)$ is an asymptotic lower bound for $f(n)$ if:

$$\Omega(h(n)) = \{f(n) : \exists \quad c, n_0 > 0 \quad s.t. \quad 0 \leq c \cdot h(n) \leq f(n), \forall n \geq n_o\} \tag{20.3}$$

PROOF **Find Θ for $f(n) \sum_i^n i$.**
For this we will employ a technique for the proof where we take the right half of the function, i.e. from $\frac{n}{2} \ldots n$ and then find the bound

$$\begin{aligned} f(n) &= 1 + 2 + 3 \ldots + n \\ &\geq \lceil \frac{n}{2} \rceil + (\left\lceil \frac{n}{2} \right\rceil + 1) + (\left\lceil \frac{n}{2} \right\rceil + 2) + \ldots n \quad n/2 \text{ times} \\ &\geq \left\lceil \frac{n}{2} \right\rceil + \left\lceil \frac{n}{2} \right\rceil + \left\lceil \frac{n}{2} \right\rceil + \ldots \left\lceil \frac{n}{2} \right\rceil \\ &\geq \frac{n}{2} \cdot \frac{n}{2} \\ &= \frac{n^2}{4} \end{aligned} \tag{20.4}$$

And therefore for $c = \frac{1}{4}$ and $n = 1$, $f(n) \in \Theta(n^2)$

□

Definition 36 Big Θ notation (asymptotically tight bound)

$$\Theta(g(n)) = \{f(n) : \exists \quad c_1 c_2, n_0 \quad s.t. \quad 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n), \forall n \geq n_o\} \tag{20.5}$$

PROOF Prove that

$$\sum_{j=1}^{n} i^k = \Theta(n^{k+1}) \tag{20.6}$$

First, prove $O(f(n)) = O(n^{k+1})$

$$\begin{aligned} f(n) = \sum_{j=1}^{n} i^k &= 1^k + 2^k + \ldots n^k \\ &\leq n^k + n^k + \ldots n^k \\ &= n^{k+1} \end{aligned} \tag{20.7}$$

Next, prove $\Omega(f(n)) = \Omega(n^{k+1})$

$$f(n) = \sum_{j=1}^{n} i^k = 1^k + 2^k + \dots n^k$$
$$= n^k + (n_1)^k + \dots 2^k + 1^k = \sum_{i=1}^{n} (n-i+1)^k \tag{20.8}$$
$$\geq \frac{n}{2}^k * n \geq \frac{n^{k+1}}{2^k} = \Omega(n^{k+1})$$

Therefore $f(n) = \Theta(n^{k+1})$ □

Note that we may not always find both a tight upper and lower bound so not all functions have a tight asymptotic bound.

Theorem 9 **Properties of asymptotes:**
Note: $\wedge$ means AND
Transitivity [23]

$$(f(n) = \Theta(g(n)) \wedge g(n) = \Theta(h(n))) \Rightarrow f(n) = \Theta(h(n)) \tag{20.9}$$

Reflexivity[24]

$$f(n) = \Theta(f(n)) \tag{20.10}$$

Symmetry

$$f(n) = \Theta(g(n)) \iff g(n) = \Theta(f(n)) \tag{20.11}$$

Transpose Symmetry

$$\begin{aligned} f(n) = O(g(n)) &\iff g(n) = \Omega(f(n)) \\ f(n) = o(g(n)) &\iff g(n) = \omega(f(n)) \end{aligned} \tag{20.12}$$

[23] *The following applies to O, Θ, o, ω*

[24] *The following applies to O, Θ*

Runtime complexity bounds can sometimes be used to compare functions. For example, $f(n) = O(g(n))$ is like $a \leq b$

- $O \approx \leq$
- $\Omega \approx \geq$
- $\Theta \approx \approx$
- $o \approx <$; an upper bound that is **not** asymptotically tight
- $\omega >$ a lower bound that is **not** asymptotically tight

Note that there is no trichotomy; unlike real numbers where we can just do $a < b$, etc, we may not always be able to compare functions.

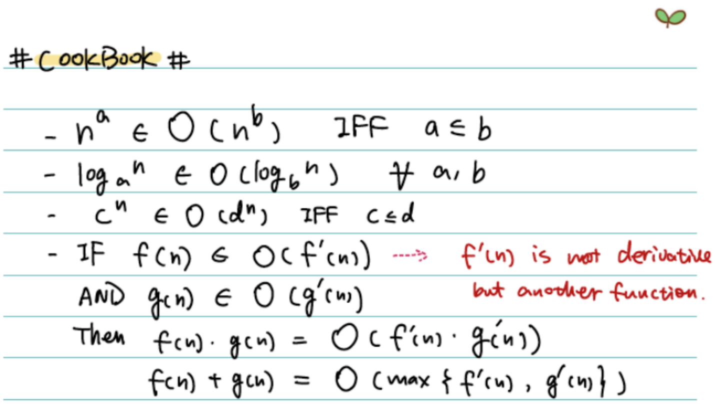


Figure 24. Complexity Cookbook

Subsection 20.2

Lecture 3: Logs & Sums

20.2.1 Functional Iteration

Recall:

$$a = b^c \Leftrightarrow log_b a = c \quad (20.13)$$

$f^{(i)}(n)$ denotes a function iteratively applied i times to value n.

For example, a function may be defined as:

$$f^{(i)}(n) = \begin{cases} f(n) & \text{if } i = 0 \\ f(f^{(i-1)}(n)) & \text{if } i > 0 \end{cases} \quad (20.14)$$

Given (20.14) we see that

1. $f(n) = 2n$
2. $f^{(2)}(n) = f(2n) = 2^2 n$
3. $f^{(3}(n) = f(f^{(2)}(n)) = 2^3 n \vdots$
4. $f^{(i)}(n) = 2^i n$

As an exercise we may look at an iterated logarithm function, 'log star'

$$lg^*(n) = \min\{i \geq 0 : lg^{(i)} n \leq 1\} \quad (20.15)$$

This describes the number of times we can iterate $log(n)$ until it gets to 1 or smaller.

- $log^* 2 = 1$
- $log^* 4 = 2 = log^* 2^2 = 1 + log^* 2 = 2 \vdots$
- for practical reasons log^* doesn't really get bigger than 5. This is one of the slowest growing functions around.

Summations & Series

Proof | Proof for a finite geometric sum:

$$\sum_{k=0}^{n} x^k = S$$

$$\begin{aligned} S &= 1 + x + x^2 \dots x^n \\ xS &= x + x^2 + x^3 \dots x^{n+1} \\ S &= \frac{1 - x^{n+1}}{1 - x} \end{aligned} \tag{20.16}$$

□

$$\sum_{i=1}^{\infty} x^i = \frac{1}{1-x} \quad \text{if } |x| < 1 \tag{20.17}$$

$$\sum_{k=0}^{\infty} kx^k = \frac{x}{(1-x)^2} \quad \text{if } |x| < 1 \tag{20.18}$$

PROOF Begin by differentiating both sides over x

$$\sum_{k=0}^{\infty} x^k = \frac{1}{(1-x)} \quad \text{if } |x| < 1 \tag{20.19}$$

$$\sum_{k=0}^{\infty} kx^{k-1} = \frac{1}{(1-x)^2} \tag{20.20}$$

And then multiply both sides by x, therefore (20.18) follows. □

Telescoping Series

$$\sum_{k=1}^{n} a_k - a_{k-1} = a_n - a_0 \tag{20.21}$$

PROOF Write it out and cancel out terms

$$(a_1 - a_0) + (a_2 - a_1) \dots (a_n - a_{n-1}) = a_n - a_0 \tag{20.22}$$

Therefore the sum telescopes □

Another telescoping series may be proved similarly:

$$\sum_{k=1}^{n-1} \frac{1}{k(k+1)} \xrightarrow{math} \sum_{k=1}^{n-1} (\frac{1}{k} - \frac{1}{k+1}) = (1 - \frac{1}{2}) + \dots (\frac{1}{n-1} - \frac{1}{n}) = a_o - a_n \tag{20.23}$$

SUBSECTION 20.3

Lecture 4: Induction & Contradiction

20.3.1 Induction

The general steps for proving a statement by induction are:

1. Basis
2. Hypothesis
3. Inductive step

I.e. if the basis holds for some i, i.e. $0, 1, 2, 3, 12, \ldots$ AND if we assume that the hypothesis holds for an arbitrary number k, then we just need to prove that the inductive step follows, or that $P(n+1)$ holds.

Example Prove that $P(n) = 1 + 2 + 3 + \ldots n = \frac{n(n+1)}{2}$

PROOF

1. Basis: $P(0) = 0 = \frac{0(0+1)}{2}$
2. Hypothesis (assume that it is true): $P(k) = \frac{k(k+1)}{2}$
3. Inductive step (need to prove $P(k+1) = \frac{(n+1)(n+2)}{2}$): $P(k+1) = \underbrace{1 + 2 + \ldots + n}_{\frac{n(n+1)}{2}} + (n+1) = \ldots = \frac{n^2+3n+2}{2} = \frac{(n+1)(n+2)}{2}$

□

Example Show that for any finite set S, the power set 2^S has $2^{|S|}$ elements (that is, there are $2^{|S|}$ distinct subsets of S)

The power set of a set S is the set of all subsets of S

PROOF

1. Basis:
$$n = 0, |S| = 0, |2^S| = 1 = 2^0 \tag{20.24}$$
$$n = 1, |S| = 1, |2^S| = 2 = 2^1 \tag{20.25}$$
2. Hypothesis: Assume that 2^S has 2^n elements when $|S| = n$
3. Inductive step: need to prove that when $|S| = n+1, |2^S| = 2^{n+1}$

   Let $B = S \not\{a\}$ for some $a \in S$. Now there are two types of subsets of S; those that include a and those who do not include a

   For subsets that do *not* include a , $|2^B| = 2^{|B|} = 2^n$, by the hypothesis.

   For subsets that do include a, these sets are of size $2^B \cup \{a\}, which is 2^n$.

   Therefore the total number of subsets of S is $2^n + 2^n = 2^{n+1}$, as desired.

□

The same kind of argument can be applied to problems such as the Towers of Hanoi and the tiling problem.

20.3.2 Contradiction

1. Assume the theorem is false
2. Show that the assumption is false (leads to a contradiction)
   - Therefore the theorem is true

Example Prove that $\sqrt{2}$ is irrational

Proof Assume that $\sqrt{2}$ is rational.
Therefore we can write $\sqrt{2}$ as

$$\sqrt{2} = \frac{a}{b} \tag{20.26}$$

Where a, b **have no common factors**.
We can square both sides

$$2 = \frac{a^2}{b^2} \to a^2 = 2b^2 \tag{20.27}$$

Therefore a^2 is even.
Let $a = 2c$

$$2^2c^2 = 2b^2 \to b^2 = 2c^2 \tag{20.28}$$

Therefore b is even as well.
This results in a contradiction since we assumed that a, b have no common factors, but our analysis shows that both would have to be even (and share a common factor of 2). □

Subsection 20.4

Lecture 5: recurrences

Many recursive algorithms can be thought of as a divide-and-conquer approach where we break the problem into subproblems that are similar to the origianl but smaller in size, solve them resurisively, then combine them to create a solution to the original problem.

Definition 37 A recurrence is a function defined in terms of:

- 1+ base cases
- Itself, with smaller arguments

For example, finding a Fibonacci number is a recurrence;
$T(n) = T(n-1) + T(n-2)$ with some base cases.

Example **Mergesort**
Sorting $[3, 1, 7, 5]$

1. Divide: break into partitions: $[[3, 1], [7, 5]]$
2. Sort partitions: $[[1, 3], [5, 7]]$
3. Create result array
4. Compare: have two pointers to front of array
   - compare 1, 5. 1 is smaller; $result = [] \leftarrow 1$
   - Move ptr to left array (1, 3) ahead one. Compare 3, 5. 3 is smaller, so $result = [3] \leftarrow 3$
   - One of the arrays is now empty so we can just append the rest
5. $result = [1, 3, 5, 7]$

Definition 38 Pseudocode for mergesort is given by:

```
mergesort(A, p, r)
        if p < r
                q = [(p+r)/2]
                mergesort(A, p, q)  // N/2
                mergesort(A, q+1, r)  // N/2
                merge(A, p, q, r) // merge the sorted
                 ↪  subarrays
```

This mergesort partitions in half each time[25].
In the worst case we will compare $N-1$ times, so $O(N)$ worst case.
Proving that merge sort is $\Omega(N)$

[25] *binary partitioning(?)*

Here we're discussing not time complexity but rather the number of times we call mergesort

How much time does MergeSort take?
The time of mergesort is defined recursively as:

$$T(N) = \begin{cases} O(1) & n = 1 \\ T(N) = 2T(\frac{N}{2} + \Theta(N) & \end{cases} \tag{20.29}$$

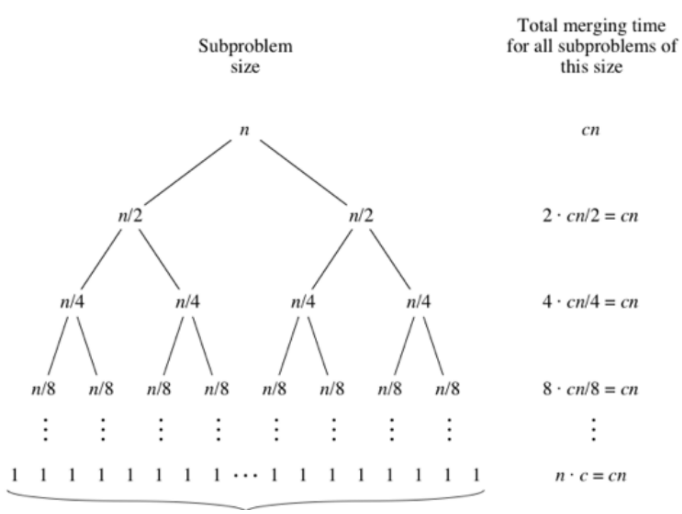
More generally we can find the runtime of a recurrence algorithm via

- Recurrence trees
- Substitution
- Master theorem

Note that $\Theta(N)$ is for the merge operation

20.4.1 Recurrence Trees

Example Recurrence trees can be used to find the time complexity of mergesort.



The height of the tree is $\log N$
The total cost is the total cost per level times the number of levels, which is

$$N \cdot logN \tag{20.30}$$

So the complexity is $O(N \log N)$

20.4.2 Substitution

1. Guess the answer
2. Apply induction

Example Determine an asymptotic upper bound on $T(n) = 2T(\lfloor \frac{n}{2} \rfloor) + \Theta(n)$

Proof This expression can be simplified since we don't care too much about floors or ceilings for asymptotic behaviour.

$$T(n) = 2T(\frac{n}{2}) + N \tag{20.31}$$

Let's guess that the upper bound is $O(n \log n)$
Then, we need to prove that $T(n) < C \cdot n \log n$ for some $C > 0$. Let's apply induction.

1. Basis: this is tricky since if $n = 1$ we end up with $T(1) \leq C \cdot 1 \cdot \log 1 = 0$ which cannot hold since that would just not make sense. Instead, observe that $T(1) = 1$, $T(2) = 2T(1) + 2 = 4, T(3) = 2T(1) + 3 = 5, T(4) = 2T(2) + 4 \ldots$.

   So $T(n)$ is therefore independent of $T(1)$, so we can use two bases, $T(2), T(3)$. Since $T(2) \leq C * 2 \log 2 = 2C, T(3) \leq C \cdot \log$

2. Hypothesis: Assume that the upper bound holds for all possible $m < n$, let $m = \lfloor \frac{n}{2} \rfloor$. This yields $T(\frac{n}{2}) \leq C \cdot \lfloor \frac{n}{2} \rfloor \cdot \log \lfloor \frac{n}{2} \rfloor$

3. Inductive step: substitute hypothesis into recurrence yields

$$T(N) \leq C \cdot (C \cdot \left\lfloor \frac{N}{2} \right\rfloor \cdot \log \left\lfloor \frac{N}{2} \right\rfloor + N = cN \log N - (1-c)N \leq Cn \log n \tag{20.32}$$

□

A few pitfalls to avoid is guessing $T(n) = O(n) = c \cdot n$ and so forth we would get

$$T(N) \leq 2C \cdot \left\lfloor \frac{n}{2} \right\rfloor + n = cn + n = (c+1)n \tag{20.33}$$

This would be wrong since we cannot change the constant to $c+1$; we have to prove it with exactly the hypothesis given.

20.4.3 Master Theorem

Definition 39 The master method applies to recurrences of the form

$$T(n) = aT(\frac{n}{b}) + f(n) \qquad \text{where } a \geq 1, b > 1, f \text{asymptotically positive} \tag{20.34}$$

It distinguishes 3 common cases b comparing $f(n)$ with $n^{\log_b a}$

1. If $f(n) = O(n^{\log_b a - \varepsilon})$ for some $\varepsilon > 0$, then $T(n) = \Theta(n^{log_b a})$
2. If $f(n) = \Theta(n^{\log_b a})$ for some $\varepsilon > 0$, then $T(n) = \Theta(n^{log_b a} \log n)$
3. If $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some $\varepsilon > 0$ and $af(\frac{n}{b}) \leq cf(n)$ for some $c < 1$, then then $T(n) = \Theta(f(n))$

Proof is out of scope for the course

There are a few technicalities to be a ware of. In each example we compare $f(n)$ with $F = n^{\log_b a}$ and take the larger of each as the solution the recurrence. For the first case we note that $f(n)$ must be *polynomially* smaller than F; i.e. it must be asymptotically smaller than F by some factor of n^{ε}. In the third case $f(n)$ must be greater than F as well as being polynomially larger and satisfy the regularity condition $af(\frac{n}{b}) \leq cf(n)$. There are areas where the master theorem does not cover, for example a gap between cases 1, 2 where $f(n) > F$ but is not polynomially larger. If $f(n)$ falls in one of these gaps or the regularity condition does not hold, the master method cannot be used to solve the recurrence.

Example What is the closed form of $T(n) = T(\frac{2n}{3}) + 1$?

PROOF a = 1, b = 2/3, f(n) = 1.

$$\log_b a = \log_{\frac{2}{3}} 1 = 0 \tag{20.35}$$

$$f(n) = \Theta(n^0) \tag{20.36}$$

So

$$T(n) = O\log(n) \tag{20.37}$$

□

SUBSECTION 20.5

Lecture 6

20.5.1 Graphs

Definition 40 A **graph** is a data structure comprised from set of vertices V and a set of edges E, where each edge connects a pair of vertices. A **directed graph (digraph)** is a graph where edges E have a *direction*, i.e. an edge (u, v) is different from (v, u). Conversely, an **undirected graph** is a graph where edges E do not have a direction.

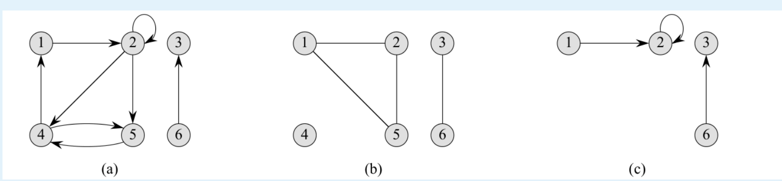


Figure 25. (a) directed graph, (b) undirected graph, (c) a subgraph of (a)

Some conventions:

- Edges are denoted by (u, v)[26] where $u, v \in V$
- If (u, v) is a edge in a directed graph, then (u, v) is incident from or leaves u, and is incident to or enters v
- If (u, v) is a edge a graph, then u , v are adjacent
- The **degree** of a vertex is the number of edges incident to it
- A **path** is a sequence of vertices $(v_0, v_1, \ldots v_k)$ from from vertex to another such that each vertex is incident[27] to the ones prior and after.
  - If there exists a path from a to b then b is **reachable** from a and a
  - A path is **simple** if no vertex is repeated
  - A path forms a cycle if the first and last vertices are the same

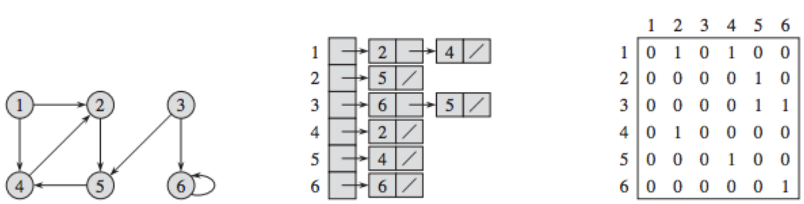
[26] *not* $\{u, v\}$

[27] *with the exception of start/end vertices*

- A directed graph with no self-loops is **simple**
- A graph with no simple cycles is acyclic
- An undirected graph is **connected** if there exists a path between any two vertices
- A directed graph is **strongly connected** if every vertex is reachable from every other vertex
- Two graphs V, V' are **isomorphic** if there exists a bijection[28] between the vertices of the two graphs such that the edges are preserved
- Given graph G, $G' = (V', E')$ is a **subgraph** of G if $V' \subseteq V$ and $E' \subseteq E$
- Given a set $V' \subseteq V$, the subgraph of G **induced** by V' is the graph $G' = (V', E')$ where $E' = \{(u, v) \in E | u, v \in V'\}$
- Given an undirected graph, the **directed version** of G is $G = (V, E')$ where $(u, v) \in E'$ if and only if $(u, v) \in E$. In other words we replace all undirected edges and replace them with their directed counterpart.
- The corollary can be applied to a directed graph to get the **undirected version** of G.
- A **neighbor** of u in a directed graph is any vertex v such that $(u, v) \in E$ where E is the set of edges for the undirected counterpart of the graph
- A **complete** graph is a graph where every pair of vertices are connected by an edge
- A **bipartite graph** is an undirected graph G in which it's V can be partitioned into two disjoint sets V_1, V_2 such that every edge $(u, v) \in E$ connects a vertex in V_1 to a vertex in V_2 or vice-versa
- An acyclic undirected graph is a **forest**
- A connected acyclic undirected graph is a **free tree**.
  - A directed acyclic graph is often termed a DAG
- A multi-graph is a graph where edges can be repeated and self-loops are allowed
- A hyper-graph is a graph where edges can connect more than two vertices
- The **contraction** of an undirected graph G by an edge $e = (u, v)$ is a graph G' where $V' = V - \{u, v\} \cup \{x\}$, where x is a new vertex. Then, for each edge connected to u, v the edges are deleted and then reconstructed with x, effectively 'contracting' u, v into a single vertex

[28] *$f : V \to V'$, i.e. we can relabel the vertices of V to be those of V' and the two graphs would be identical*

In code graphs are commonly represented as adjacency lists or adjacency matrices. This was covered in ESC190, but for reference:



(a) (b) (c)

Table 5. Time complexities of graph representations

	adjacency list	matrix
Time	$O(n)$	$O(1)$
Memory	$O(E)$	$O(n^2)$

20.5.2 Trees

A **tree** is a common and useful subset of graphs

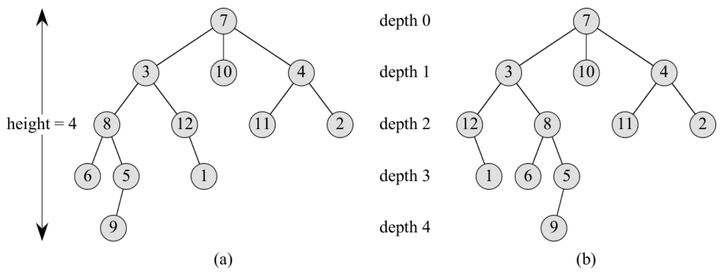


Figure B.6 Rooted and ordered trees. **(a)** A rooted tree with height 4. The tree is drawn in a standard way: the root (node 7) is at the top, its children (nodes with depth 1) are beneath it, their children (nodes with depth 2) are beneath them, and so forth. If the tree is ordered, the relative left-to-right order of the children of a node matters; otherwise it doesn't. **(b)** Another rooted tree. As a rooted tree, it is identical to the tree in (a), but as an ordered tree it is different, since the children of node 3 appear in a different order.

Definition 41 A tree is a common subset of graphs, i.e. ones that are **connected, acyclic, and undirected**. This gives a few useful properties, i.e. the existence of a *root* node, the parent-child relationship, and the existence of a unique path between any two nodes.

Some conventions for trees

- Depth of node: length from root to node
- Height length of longest path from node to leaf
- Degree of node: number of children. Binary trees have degree 2, n-ary tree has degree n

Theorem 10 All of the following statements are equivalent for a tree $T = (V, E)$:

1. $\forall v \in V, v$ is a tree; all nodes in a tree are trees unto themselves.
2. Every two nodes are connected by a unique path
3. T is connected by if any edge is removed the resulting graph is disconnected T is connected and $|E| = |V| - 1$ T is acyclic and $|E| = |V| - 1$ T is connected but if a edge is added the resulting graph has a cycle

Subsection 20.6

Lecture 7: Probability and Counting

Most of the probability stuff is review from ECE286 so I'll be omitting most notes.

Definition 42 Probability distribution $Pr\{\}$: mapping from events of S to real numbers where

1. $Pr\{\emptyset\} = 0$

2. $Pr\{A\} \geq 0 \forall A \in S$
3. $Pr\{S\} = 1$
4. $Pr\{A \cup B\} = Pr\{A\} + Pr\{B\} \forall A, B \in S$

For any two events A, B we can define the triangle inequality

The complement of an event A is $\overline{A} = S - A$, and the probability is $Pr\left\{\overline{A}\right\} = 1 - Pr\{A\}$

Definition 43 $Pr\{A \cup B\} \leq Pr\{A\} + Pr\{B\}$

Definition 44 Baye's thereom

$$Pr\{A \mid B\} = \frac{Pr\{B \mid A\} Pr\{A\}}{Pr\{B\}} = \frac{Pr\{A\} Pr\{B|A\}}{Pr\{A\} Pr\{B|A\} + Pr\left\{\overline{A}\right\} Pr\left\{B|\overline{A}\right\}} \tag{20.38}$$

The expected value of a random variable is

$$E[X] = \int_{-\infty}^{\infty} x Pr\{X = x\}\, dx \tag{20.39}$$

And in the discrete case,

$$E[X] = \sum_{x \in \mathbb{Z}} x Pr\{X = x\} \tag{20.40}$$

The variance is

$$Var[X] = E\left\{(X - E[X])^2\right\} = E[X^2] - E^2[X] \tag{20.41}$$

PART

VI

MAT389: Complex Analysis

Taught by Prof. Sigil

Section 21

Complex Numbers

Subsection 21.1

Lecture 1

Consider a 2-vector $\vec{x} = (x, y) \in \mathcal{R}$. As complex numbers correspond to two-vectors

$$\vec{x} = (x, y) \leftrightarrow z = x + iy, i^2 = -1 \tag{21.1}$$

z is, therefore, a complex variable. What are the benefits of a complex number like z?

This prof lectures at the speed of sound and talks *into* the board. Couldn't quite follow during this lecture, hopefully I get better about it in the following ones.

Definition 45 **Imaginary and Complex Numbers**
i is an imaginary number such that

$$i^2 = -1 \tag{21.2}$$

A complex number has the form:

$$z = x + iy \tag{21.3}$$

Definition 46 There are a number of operations we can perform on complex numbers.
Addition

$$z + z' = (x + x') + i(y + y') \tag{21.4}$$

Multiplication

$$zz' = (x + iy)(x' + iy') = (xx' - yy') + i(xy' + x'y) \tag{21.5}$$

Proof Proof of (21.5):

$$\begin{aligned} zz' &= (x + iy)(x' + iy') \\ &= x + ixy' + iyx' + i^2yy' \\ &= xx' - yy' + i(xy' + yx') \end{aligned} \tag{21.6}$$

□

Magnitude

$$|z| = \sqrt{x^2 + y^2} \tag{21.7}$$

Conjugate
The complex conjugate has the properties:

- $\overline{z}z = |z|^2$
- $\overline{(z + z')} = \overline{z} + \overline{z}'$
- $\overline{z \cdot z'} = \overline{z} \cdot \overline{z}'$

We can define a new operation

$$\forall \text{complex} z, \exists \quad \text{complementary number } w \text{ such that } zw = wz = 1 \tag{21.8}$$

Denote

$$w = \frac{1}{z} = z^{-1} \tag{21.9}$$

Proof Proof of (21.9): Find w s.t. $zw = 1$

$$\begin{aligned} zw &= 1 \\ w\overline{z}z &= \overline{z}z = |z|^2 > 0 \\ |z|^2 w &= \overline{z} \\ w &= \frac{\overline{z}}{|z|^2} \to Z^{-1} = \frac{\overline{z}}{|z|^2} \end{aligned} \tag{21.10}$$

□

Furthermore, there are operators that we can define on complex numbers.

Definition 47 **Real and Imaginary Operators**
Given $z = x + iy$, we can define the real and imaginary operators

$$x = Re\{z\} \tag{21.11}$$

$$y = Im\{z\} \tag{21.12}$$

Example

$$Im\left\{(1 = \sqrt{2}i)^{-1}\right\} \tag{21.13}$$

By (21.9), we have

$$Im\{z^{-1}\} = \frac{-Im\{z\}}{|z|^2} \tag{21.14}$$

And

$$Re\{z^{-1}\} = \frac{-Re\{z\}}{|z|^2} \tag{21.15}$$

Using these, for example, we find that the $Im = \frac{-\sqrt{2}}{3}$
We can get the real component in a similar way.

Here is an enumeration of absolute value properties for complex numbers:

$$|z \cdot w| = |z||w| \tag{21.16}$$

$$|z + w| \leq |z| + |w| \tag{21.17}$$

$$|\overline{z}| = |z| \tag{21.18}$$

$$|z + w|^2 = (\overline{x} + \overline{w})(z + w) = |z|^2 + |w|^2 + \overline{z}w + \overline{w}z \tag{21.19}$$

Proof. Note that $\overline{z}w + \overline{w}z = 2Re\{z\overline{w}\}$, by (21.19)
And so

$$|z + w|^2 \leq |z|^2 + |w|^2 + 2|z||w| = (|z| + |w|)^2 \tag{21.20}$$

□

Subsection 21.2
Lecture 2

Whereas a two-vector $\vec{x} \in \mathbb{Z}$, complex numbers exist in the complex plane, $z \in \mathbb{C}$

Theorem 11 **Polar Decomposition**
Complex numbers can be expressed in polar form as well

$$z = r(\cos\theta + i\sin\theta) \tag{21.21}$$

Where

$$r = |z| \qquad x = r\cos\theta \qquad y = r\sin\theta \qquad \theta = \tan^{-1}\left(\frac{y}{x}\right) \tag{21.22}$$

This has a number of useful properties

$$z \cdot z' = |z||z'|(\cos(\theta + \theta') + i\sin(\theta + \theta')) \tag{21.23}$$

$$\frac{z}{z'} = \frac{|z|}{|z'|}(\cos(\theta - \theta') + i\sin(\theta - \theta')) \tag{21.24}$$

Proof: Proof for (21.23):

$$\begin{aligned} z \cdot z' &= |z|(\cos(\theta + i\sin\theta)) \times |z'|(\cos\theta' + i\sin\theta') \\ &= |z||z'|(\cos\theta\cos\theta' + i\cos\theta\sin\theta' + i\sin\theta\cos\theta' - \sin\theta\sin\theta') \\ &= |z||z'|\left[\cos\theta\cos\theta' - \sin\theta\sin\theta' + i(\cos\theta\sin\theta' + \sin\theta\cos\theta')\right] \end{aligned} \tag{21.25}$$

And the proof follows □

Lemma 3 A corollary exists

$$z^2 = |z|^2(\cos 2\theta + i\sin 2\theta) \tag{21.26}$$

Theorem 12 **Moivre's Theorem**

$$z^n = |z|^n(\cos(n\theta) + i\sin(n\theta)) \tag{21.27}$$

More generally,

$$(\cos\theta + i\sin\theta)^n = \cos(n\theta) + i\sin(n\theta) \tag{21.28}$$

So we may define z to be the n^{th} root of w which implies that

Lemma 4 Every complex number has a n^{th} root $\forall n$

Proof:

$$\text{Let } z = |w|^{\frac{1}{n}}(\cos\frac{\theta}{n} + i\sin\frac{\theta}{n}) \tag{21.29}$$

Then

$$w = |w|(\cos\theta + i\sin\theta), \text{ then } z^n = w \tag{21.30}$$

□

Intuition: define z to be $\frac{1}{n}$ and then take the n^{th} power of both sides to show that $z^n = w$

This leads us to the conclusion that representations of complex numbers are not unique[29].

[29] *They are part of a cyclic group*

Proof: If every z can be written as $z = r(\cos\theta + i\sin\theta)$, then it holds for $\theta + 2\pi n \forall n \in \mathbb{Z}$ since $\sin\theta = \sin(\theta + 2\pi n)$ and $\cos\theta = \cos(\theta + 2\pi n)$. □

21.2.1 Functions on complex planes

Definition 48 Given a domain $\mathbb{D} \in \mathbb{C}$, a function f is a rule such that

$$z \in \mathbb{D} \xrightarrow{f} w \in \mathbb{D} \leftrightarrow w = f(z) \tag{21.31}$$

Definition 49 We may define $\mathbb{D}$ to be the domain of f
Likewise, range is defined as

$$Ran\{f\} = \{w \in \mathbb{C} : \exists z \in D : f(z) = w\} \tag{21.32}$$

Example

$$f(z) = \frac{1}{z+i} \tag{21.33}$$

What is the maximum domain of f?

$$Dom\{f\} = \{z \in \mathbb{C} : |z| < -i\} \tag{21.34}$$

What is the range of f?

$$\frac{1}{z+i} = w \tag{21.35}$$

For which values of w can we solve this equation?

$$z = -i + \frac{1}{w} \tag{21.36}$$

So the range of the function is

$$Ran\{f\} = \{w \in \mathbb{C} : |w| \neq 0\} \tag{21.37}$$

Example

$$f(z) = z^2 + 1 \tag{21.38}$$

It is fairly clear that $Dom\{f\} = f \in \mathbb{C}$
The range can be found by solving for z in

$$z^2 + 1 = w \tag{21.39}$$

And so

$$Ran\{f\} = \{w \in \mathbb{C}\} \tag{21.40}$$

21.2.2 Exponential Functions

Definition 50 Given $z = x + iy$,

$$e^z = e^x(\cos y + i \sin y) = e^{Re\{z\}}(cos(Im\{z\}) + i\sin(Im\{z\})) \tag{21.41}$$

1. $e^{z+w} = e^z e^w$
2. $|e^z| = e^{Re\{z\}} \neq 0$
3. $e^{z+i2\pi n} = e^z$

PROOF (1) follows from the product rule for complex numbers
(2) follows by definition
(3) follows by definition (recall: writing z w.r.t. sin, cos) □

More properties:

arg, or argument is the angle from the real axis to that on the complex plane. Usually denoted by θ

- $Dom\{e^z\} = \mathbb{C}$
- $Ran\{e^z\} = \{\mathbb{C} \setminus \{0\}\}$ [30]
- $e^z = w \qquad$ if $w \neq 0$

[30] Note: ' \ ' denotes set exclusion

$$
\begin{aligned}
z &= ln|w| + i \arg w \\
e^z &= e^{ln|w| + i \arg w} \\
&= e^{ln|w|} e^{i \arg w} \\
&= |w| \cos(\arg w) + i \sin(\arg w) \\
&= w
\end{aligned}
\tag{21.42}
$$

Remark **Polar representation**

$$w = |w| e^{i \arg w} \tag{21.43}$$

$$r = |w| \quad \theta = \arg w \tag{21.44}$$

Example **Find polar coordinates of $z = i + 1$**

$$
\begin{aligned}
|z| &= \sqrt{1+i} = \sqrt{2} \\
\cos\theta &= \frac{1}{\sqrt{2}} \to \theta = \frac{\pi}{4} \\
z &= \sqrt{2} e^{i\pi/4}
\end{aligned}
\tag{21.45}
$$

Example Find

$$(1+i)^{\frac{1}{3}} \tag{21.46}$$

Solution: $z = \sqrt{2} e^{\frac{i\pi}{4}} \to z^{1/3} = 2^{\frac{1}{6}} e^{i\pi/12}$

Definition 51 **Trig functions for complex numbers**

$$\cos z = \frac{1}{2}\left(e^{iz} + e^{-iz}\right) \tag{21.47}$$

Proof

$$\cos x = \frac{1}{2}(e^{ix} + e^{-ix}) = \frac{1}{2}\left(\cos x + i \sin x + \underbrace{\cos(-x)}_{\text{odd; } = \cos(x)} + \underbrace{i \sin(-x)}_{\text{even; } = -\sin(x)}\right) = \cos x \tag{21.48}$$

□

$$\sin z = \frac{1}{2}\left(e^{iz} - e^{-iz}\right) \tag{21.49}$$

And a similar proof follows for $\sin z$.
These have the following properties

$$\sin z|_{ImZ=0} = \sin x \tag{21.50}$$

$$\cos(z + 2\pi n) = \cos z \forall n \in \mathbb{Z} \tag{21.51}$$

$$\sin(z + 2\pi n) = \sin z \forall n \in \mathbb{Z} \tag{21.52}$$

Proof

$$\begin{aligned} \cos z + 2\pi n &= \frac{1}{2}(e^{i(z+2\pi n)} + e^{-i(z+2\pi n)}) \\ &= \frac{1}{2}(e^{iz}e^{i2\pi n} + e^{-iz}e^{-i2\pi n}) \\ &= \frac{1}{2}(e^{iz} + e^{-iz}) \\ &= \cos z \end{aligned} \tag{21.53}$$

□

The domain of $\{\cos z, \sin z\} = \mathbb{C}$
Range?
Solve $\cos z = w$ for z

$$\begin{aligned} \frac{1}{2}(e^{iz} + e^{-iz}) &= w \\ \ldots \times 2e^{iz} &\text{ on both sides} \\ e^{2iz} - 2we^{iz} + 1 &= 0 \\ &\ldots\text{Let } S = e^{iz} \\ S^2 - 2ws + 1 &= 0 \\ &S = w \pm \sqrt{w^2 - 1} \equiv u \end{aligned} \tag{21.54}$$

Now we note that $e^{iz} = u$ can be solved for z for any $u \neq 0$

$$u = 0 \leftrightarrow w = \pm\sqrt{w^2 - 1} \tag{21.55}$$

$$w^2 = w^2 - 1 \text{ impossible for } u \neq 0 \tag{21.56}$$

Therefore:

$$Ran\{\cos z\} = Ran\{\sin z\} = \mathbb{C} \tag{21.57}$$

Remark An intuitive way of interpreting this result is thinking of $\{\sin, \cos\}$ being a function that projects values from the complex domain to the real plane; though $\{\sin, \cos\}$ takes on a limited range of values in the real domain, in the complex domain it spans the entire plane. Think: mental model of a complex number spinning around and having that project onto a real line. More formally, see: the Little Picard Theorem

Subsection 21.3

Lecture 3: Exponent and Logarithm

21.3.1 Exponential

Recall: the complex exponential function $\exp$ is defined as

$$\exp : e^z = e^x(\cos y + i \sin y) \tag{21.58}$$

Where $z = x + iy$.
Properties:

1. $e^{w+z} = e^z e^w$
2. $e^z \neq 0$

3. $e^{2\pi m i} = 1$

The first and third properties imply that the exponential function is a periodic function.

$$e^{z+2\pi m i} = e^z \tag{21.59}$$

Consider the equation

$$e^z = w \tag{21.60}$$

If this has the solution z_*, then $z_* + 2\pi m i, m = 0, \pm 1, \pm 2, \dots$ is also a solution.

21.3.2 Logarithm

Definition 52

$$\log \equiv \log w = \ln |w| + i \arg w \tag{21.61}$$

$w \neq 0$

Proof Proof that $\log w = \ln |w| + i \arg w$

$$\begin{aligned} w = &= |w| e^{i \arg w} \\ &= e^{\ln |w|} e^{i \arg w} \\ &= e^{\ln |w| + i \arg w} \\ \to e^z &= e^{\ln |w| + i \arg w} \\ \Rightarrow z &= \ln |w| + i \arg w \end{aligned} \tag{21.62}$$

□

Note that arg is a multivalued function, i.e

$$\arg w = \arg(w + 2\pi m_i), i \in \mathbb{Z}, \arg w \in [-\pi, \pi) \tag{21.63}$$

Example

$$e^z = e^5 \tag{21.64}$$

Solve for z

$$z = 5 + 2\pi m i, m \in \mathbb{Z} \tag{21.65}$$

Example Solve $e^z = i$

$$i = e^{\frac{i\pi}{2}} \tag{21.66}$$

$$|i| = 1; \arg i = \frac{\pi}{2} \tag{21.67}$$

The solution is therefore

$$z = i(\pi/2 + 2\pi m), m \in \mathbb{Z} \tag{21.68}$$

Note that providing only a single solution is wrong; must provide all

The complex logarithm is a multivalued function (like arg).

$$\log z = \ln |z| + i \arg z, \arg z \in [-\pi, \pi) \tag{21.69}$$

Note that $i \arg z$ demotes the principal branch of log.
Though it is multivalued, in general, $\log zw \neq \log z + \log w$

Example Assume $\arg z = \frac{2\pi}{3}$ and $\arg w = \frac{3\pi}{4}$.

$$\arg(zw) = ? \tag{21.70}$$

Typically we would just add them together, i.e. $\frac{2\pi}{3} + \frac{3\pi}{4}$. But this is $> \pi$ which is not allowed as per the definition of arg, so we must add or subtract something.
Let's try subtracting 2π [31]

$$\arg(zw) = \frac{17\pi}{12} - 2\pi = -\frac{7\pi}{12} \in [-\pi, \pi) \tag{21.71}$$

So we proved that in general the arguments don't sum up. But we want to go from here to proving that the logs don't sum up.

$$\begin{aligned} \log zw &= \ln|zw| + i\arg zw \\ &\neq \ln|z| + \ln|w| + i\arg z + i\arg w \end{aligned} \tag{21.72}$$

In general this is not correct because after breaking apart $\arg zw$ $\arg z$ and $\arg w$ when summed can exceed the range allowable for arg

$$\therefore \log(zw) = \log z + \log w \tag{21.73}$$

Example Compute $\log(\sqrt{3} + i)$.
Just apply the formula.

$$\log w = \ln|w| + i\arg w \tag{21.74}$$

$$\log\left(\sqrt{3} + i\right) = \ln\sqrt{4} + i\arg(\sqrt{3} + i) = \ln 2 + i(\frac{\pi}{6} + 2\pi m), m \in \mathbb{Z} \tag{21.75}$$

21.3.3 Powers

Definition 53

$$\forall a \neq 0, a^z \equiv e^{z\log a} \tag{21.76}$$

Example Complete $(1 + i)^i$

$$\begin{aligned} (1+i)^i &= e^{i\log(1+i)} \\ \log(1+i) &= \ln\sqrt{2} + i(\frac{\pi}{4} + 2\pi m), m \in \mathbb{Z} \\ \ldots &= e^{-\frac{\pi}{4} - 2\pi m} e^{i\ln\sqrt{2}} \end{aligned} \tag{21.77}$$

Subsection 21.4

Lecture 4

Definition 54 **Analytical Functions**
A complex function is a function that maps a complex variable to a complex result. A complex *analytic* function does the same thing, *and* is continuously differentiable over $\mathbb{C}$
Define $\mathbb{D}$, an open and connected subset of the complex domain $\mathbb{C}$
We define

$$f : \mathbb{D} \to \mathbb{C} \qquad \text{to be the analytic of } z_o \in \mathbb{D} \tag{21.78}$$

Now, given f, we can define the complex derivative f'

$$f'(z_0) = \lim_{z \to z_0} \frac{f(z) - f(z_0)}{z - z_0} \qquad \text{exists} \tag{21.79}$$

[31] *since adding $\pm 2\pi$ doesn't change the angle, just rotates it around once*

Noting that

$$z \to z_0 \leftrightarrow |z - z_0| \to 0 \tag{21.80}$$

(21.79) can be rewritten as

$$f'(z_0) = \lim_{h \to 0} \frac{1}{h}(f(z_0 + h) - f(z_0)) \tag{21.81}$$

$h = z - z_0; z = z_0 + h$

Example Find the complex derivative

$$f(z) = z^n \tag{21.82}$$

Proof

$$\begin{aligned} f'(z) &= \lim_{h \to 0} \frac{1}{h}\left((z+h)^n h - z^n\right) \\ &= \lim \frac{1}{h}\left(z^n = nz^{n-1}h + \binom{n}{2} z^{n-2}h^2 + \ldots + h^n - z^n\right) \\ &= \lim \frac{1}{h}\left(nz^{n-1} + \binom{n}{2} z^{n-2}h + \ldots h^{n-1}\right) \\ &\text{... Cancel out terms that go to 0} \\ &= (z^n)' \\ &= nz^{n-1} \end{aligned} \tag{21.83}$$

□

$z = x + iy, \overline{z} = x - iy$

Example Find the complex derivative

$$f(z) = \overline{z} \tag{21.84}$$

Proof

$$\begin{aligned} f'(z) &= \lim_{h \to 0} (\overline{z} + \overline{h} - \overline{z}) \\ &= \lim_{h \to 0} \frac{\overline{h}}{h} \end{aligned} \tag{21.85}$$

We then take the limit along the real and imaginary axis separately

$h = h_1 + ih_2$

Real:

$$\lim_{h \to 0} \frac{h_1}{h_1} = 1 \tag{21.86}$$

Imaginary:

$$\lim_{h \to 0} \frac{-h_2}{h_2} = -1 \tag{21.87}$$

□

So the limit (21.85) does not exist

In the previous two examples we found that z^n is analytic and $\overline{z}$ is not.

Example

$$f(z) = e^z \tag{21.88}$$

Proof

$$
\begin{aligned}
f'(z) &= f(z+h) - f(z) \\
&= e^{z+h} - e^{z} \\
&= e^{x+h_1}(\cos(y+h_2) + i\sin(y+h_2)) - e^{x}(\cos y + i \sin y)
\end{aligned}
\tag{21.89}
$$

A Taylor series can be used to expand $e^{x+h_1}, \cos(y+h_2), \sin(y+h_2)$

$$
\begin{aligned}
e^{x+h_1}cos(y+h_2)\times \\
&\left(e^{x} + e^{x}\frac{h_1}{1!} + \text{higher order terms}\right)\times \\
&\left(\cos y - \frac{\sin y}{1!}h_2 + \text{higher order terms}\right)
\end{aligned}
\tag{21.90}
$$

$$
\tag{21.91}
$$

And then a bunch of terms can be cancelled out to leave us with a couple of terms and a bunch of higher order terms in h_1, h_2

$$
= e^{x}\cos y - e^{x}\sin y + e^{x}h_1\cos y + \ldots \text{higher order terms} \tag{21.92}
$$

And as a result

$$
(e^{z})' = lim_{h\to 0}\frac{1}{h}e^{z}h = e^{z} \tag{21.93}
$$

□

So e^{z} is analytic.

21.4.1 Properties of complex derivative

1. $(f+g)' = f' + g'$
2. $(fg)' = f'g + fg'$
3. $\frac{f}{g}' = \frac{f'g - fg'}{g^2} \cdot f(g(z))' = f'(g(z))g'(z)$

For the 3rd case here, range of $g \in$ domain of f

Example

$$
(e^{z^3}) = e^{z^3}(z^3)' = 3z^2e^{z^3} \tag{21.94}
$$

Definition 55 A function f is **entire** if f is analytic in $\mathbb{C}$

Examples of entire functions include e^{z}, z^{n}. Non-analytic functions include $\frac{1}{z}$ since it is not defined at 0 i.e. it is not entire over $\mathbb{C}$

However, is $\frac{1}{z}$ analytic over the rest of $\mathbb{C}$?

Proof

$$
\begin{aligned}
\frac{1}{z}' &= \lim \frac{1}{h}\left(\frac{1}{z+h} - \frac{1}{z}\right) \\
&= \lim \frac{1}{h} - \frac{h}{(z+h)z} \\
&= -\frac{1}{z^2} - \lim - \frac{h}{(z+h)z^2} = -\frac{1}{z}
\end{aligned}
\tag{21.95}
$$

Therefore $\frac{1}{z}$ is analytic in $\mathbb{C} - \{0\}$

$\square$

Theorem 13 **Cauchy-Riemann equations**

The Cauchy-Riemann equations give us a direct way of checking if a function is differentiable and if it is, it gives us the derivative. It is a consequence of the fact that the limit defining $f(z)$ must be the same no matter what direction z is approached. Namely, if f as defined below is analytic[32]

[32] *Complex differentiable*

$$f(z) = u + iv \tag{21.96}$$

Then,

$$f'(z) = \frac{\partial u}{\partial x} + i\frac{\partial v}{\partial x} = \frac{\partial v}{\partial y} - i\frac{\partial u}{\partial y} \tag{21.97}$$

In particular we're interested in the this set of PDEs (which is called the Cauchy-Riemann equations)

$$\begin{aligned} \frac{\partial u}{\partial x} &= \frac{\partial v}{\partial y} \\ \frac{\partial u}{\partial y} &= -\frac{\partial v}{\partial x} \end{aligned} \tag{21.98}$$

The short form is as follows

$$u_x = v_y \qquad u_y = -v_x \tag{21.99}$$

If $f = u + iv$ and in $\mathbb{D}$, then u, v satisfy

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y} \tag{21.100}$$

And

$$\frac{\partial u}{\partial y} = -\frac{\partial v}{\partial x} \tag{21.101}$$

Proof Since f is analytic, then

$$f'(z) = \lim_{h\to 0} \frac{1}{h}(f(z+h) - f(z)) \tag{21.102}$$

Which exists and is independent of the way $h \to 0$.
Take the limits of the real and imaginary parts of (21.102):
Real:

$$\lim_{h_1\to 0, h_1\in\mathbb{R}} \frac{1}{h_1}(f(z+h_1) - f(z)) = \lim \frac{1}{h_1}(f(x+h_1+iy) - f(x+iy)) = \partial_x f(z) \tag{21.103}$$

$$\lim_{ih_2\to 0, h_2\in\mathbb{R}} \frac{1}{h_2}(f(z+ih_2) - f(z)) = \lim \frac{1}{ih_2}(f(x+ih_2+iy) - f(x+iy)) = -\partial_y f(z) \tag{21.104}$$

Since this limit is independent of how $h \to 0$,

$$\partial_x f(z) = -i\partial_y f(z) \tag{21.105}$$

Recall that $f = u + iv$

$$\partial_x(u+iv) = -i\partial_y(u+iv) = -i\partial_y u + \partial_y v \tag{21.106}$$

Therefore

$$\partial_x u = \partial_y v, \partial_x v = -\partial_y u \tag{21.107}$$

□

Definition 56 Complex derivative:
Using the Cauchy-Riemann equations, we can define the complex derivative of f as

$$\frac{\partial f}{\partial z}\frac{1}{2}(\frac{\partial f}{\partial x} - i\frac{\partial f}{\partial y}) \tag{21.108}$$

$$\frac{\partial f}{\partial \overline{z}}\frac{1}{2}(\frac{\partial f}{\partial x} - i\frac{\partial f}{\partial y}) \tag{21.109}$$

Then, via the Cauchy-Riemann equations, we have

$$\frac{\partial f}{\partial \overline{z}} = 0 \tag{21.110}$$

Proof Proof of (21.110):
Recall $f = u + iv$
Plug this into the LHS of the expression:

$$\frac{\partial u}{\partial x} + i(\frac{\partial v}{\partial y}) + i(\frac{\partial u}{\partial x} + i\frac{\partial v}{\partial y}) \Rightarrow \partial_x u - \partial_y v = 0, \partial_y u + \partial_x v = 0 \tag{21.111}$$

Which is the Cauchy-Riemann equations (21.107)

□

Example Is $f(z) = e^{z^5} \cdot \sin z \cdot \overline{z}^3$ analytic?

$$\frac{\partial f}{\partial z} e^{z^5} \cdot \sin z e \overline{z}^2 \neq 0 \tag{21.112}$$

So f is not analytic

Example Is $f(z) = |z|^6$ analytic?

PROOF

$$\frac{\partial y}{\partial \overline{z}} = \frac{\partial}{\partial z}(z\overline{z})^3 = z^3 \cdot 3\overline{z}^2 \neq 0 \tag{21.113}$$

□

f is not analytic.

Now that we know that analytic functions satisfy the Cauchy-Riemann equations, we can use this to prove that the converse holds, i.e that non-analytic functions do not satisfy [the Cauchy-Riemann equations]

Theorem 14 Given $f(x, y)$ continuously differentiable and satisfies the Cauchy-Riemann equations, then f is analytical

Note abbreviation higher order terms ⇔ H.O.T

PROOF

$$\begin{aligned} f(x+h_1, y+h_2) - f(x,y) &\xrightarrow{\text{Taylor series}} \\ &= f(x,y) + \partial_x f(x,y)h_1 + \partial_y f(x,y)h_2 + \text{H.O.T} - f(x,y) \\ &\dots \text{cancel terms} \dots \\ &= \partial_x = \frac{\partial f}{\partial z} + \frac{\partial f}{\partial \overline{z}} \\ &= \partial_y = \frac{\partial f}{\partial \overline{z}} - \frac{\partial f}{\partial z}\frac{1}{i} \end{aligned} \tag{21.114}$$

And then plugging this back into the initial expression we get

$$\begin{aligned} f(x+h_1, y+h_2) - f(x,y) &= (\frac{\partial f}{\partial z}\frac{\partial f}{\partial \overline{z}})h_1 + (\frac{\partial f}{\partial \overline{z}}\frac{\partial f}{\partial z})\frac{1}{i}h_2 + \text{H.O.T} \\ &= \frac{\partial f}{\partial z}(h_1 + ih_2) + \text{H.O.T} \\ &= \frac{\partial f}{\partial z}h + \text{H.O.T} \\ f'(z) &= \lim_{h\to 0}\frac{1}{h}(\frac{\partial f}{\partial z}h + \text{H.O.T}) \\ &= \frac{\partial f}{\partial z}z \text{ exists} \end{aligned} \tag{21.115}$$

So therefore f is analytic

□

Example Is $\sin z$ analytic?

$$\frac{\partial}{\partial \overline{z}}\sin z = 0 \to \sin z \text{ is analytic} \tag{21.116}$$

In 'slow motion',

$$\begin{aligned}\frac{\partial \sin z}{\partial \overline{z}} &= \frac{1}{2}\left(\frac{\partial \sin z}{\partial x} + i\frac{\partial \sin z}{\partial y}\right)\\ \sin z &= \frac{1}{2i}\left(e^{iz} - e^{-iz}\right)\\ e^{iz} &= e^{ix-y} = e^{-y}(\cos x + i\sin y)\\ \frac{\partial e^{iz}}{\partial \overline{z}} &= \frac{1}{2}(\partial_x + i\partial_y)\\ &= \ldots = 0\end{aligned} \tag{21.117}$$

Therefore $\sin z$ is analytic

TLDR of the lecture; one can find if the function is analytic by checking if the Cauchy-Riemann equations hold. This can be done by taking the complex derivative of the function w.r.t z and $\overline{z}$. Then, by the theorem we proved, if $\frac{\partial f}{\partial \overline{z}} = 0$, then f is analytic and $f'(z) = \frac{\partial f}{\partial z}$

SUBSECTION 21.5
Power series

Definition 57 A **Power series** is an expression of the form

$$\sum_{n=0}^{\infty} a_n(z-z_0)^n \tag{21.118}$$

Where a_n is a coefficient and z_0 the centre of the series. The power series diverges if it is not converging absolutely, which it does at z_* if

$$\sum |a_n|(z_* - z_0)^n \tag{21.119}$$

converges

Theorem 15 There exists radius of convergence $R \geq 0$ such that

- The series converges absolutely in the disk $|z - z_0| < R$
- The series diverges for $|z - z_0| \geq R$

If

$$\lim_{n\to\infty} |a_n|^{1/n} \tag{21.120}$$

exists, then

$$\frac{1}{R} = \lim_{n\to\infty} |a_n|^{1/n} \tag{21.121}$$

Lemma 5 If $\lim |\frac{a_{n+1}}{a_n}|$ exists, then

$$\frac{1}{R} = \lim_{n\to\infty} |a_{n+1}/a_n| \tag{21.122}$$

Example Find the radius of convergence for

$$\sum_{n=0}^{\infty} \sqrt{n} z^n \tag{21.123}$$

PROOF Let $z_0 = 0, a_n = \sqrt{n}$

$$\lim_{n\to\infty} \sqrt{n}^{\frac{1}{n}} = \lim_{n\to\infty} n^{1/2n}$$
$$\ln n^{1/2n} = \frac{1}{2n} \ln(n) \text{ which } \to 0 \text{ as } n \to \infty \tag{21.124}$$
$$n^{\frac{1}{2n}} \to e^0 = 1 \Rightarrow R = \frac{1}{1} = 1$$

□

Theorem 16 If $\sum a_n z^n$ and $\sum b_n z^n$ have radius of conversion of at least R

1. $$\sum a_n z^n + \sum b_n z^n = \sum (a_n + b_n) z^n \tag{21.125}$$
   and has radius of convergence of at least R
2. $$\sum a_n z^n \cdot \sum b_n z^n \tag{21.126}$$
   has radius of convergence equal to R
3. $$\sum a_n z^n \tag{21.127}$$
   is complex differentiable (and therefore analytic) in $|z| < R$ and it's derivative is
   $$\sum a_n n z^{n-1} \tag{21.128}$$
   with radius of convergence R
4. $$a_n = \frac{1}{n! f^n|_0} \tag{21.129}$$
   where
   $$f(z) = \sum_{n=0}^{\infty} a_n z^n \tag{21.130}$$

SUBSECTION 21.6
Lecture 5

PART VII

ECE444: Software Engineering

Taught by Prof. Shurui Zhou

SECTION 22
Preliminary

SUBSECTION 22.1
Lecture 1, 2

- Software engineering is different from what coding is; design, architecture, documentation, testing, etc v.s. just script kiddie-ing
- Vasa syndrome
- Rockstar engineers are a myth

Section 23
Project Management

Subsection 23.1
Lecture 3

Definition 58 Conway's law states that 'Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure'.

The waterfall method is slow and costly and defects can be extremely costly, especially early on in the development lifecycle.

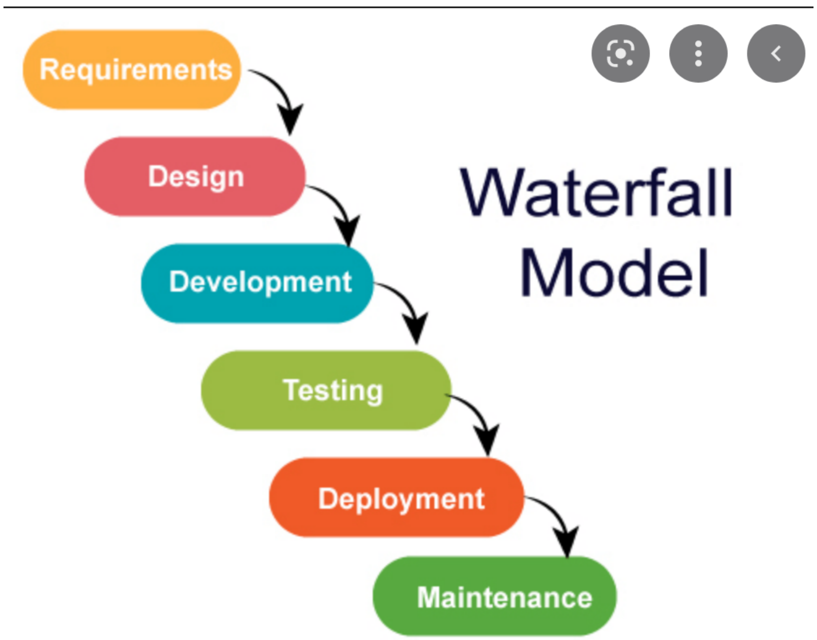


Figure 26. Waterfall method

In order to address this the V model was introduced which increases the amount of testing to reduce the possibility of having to rework everything

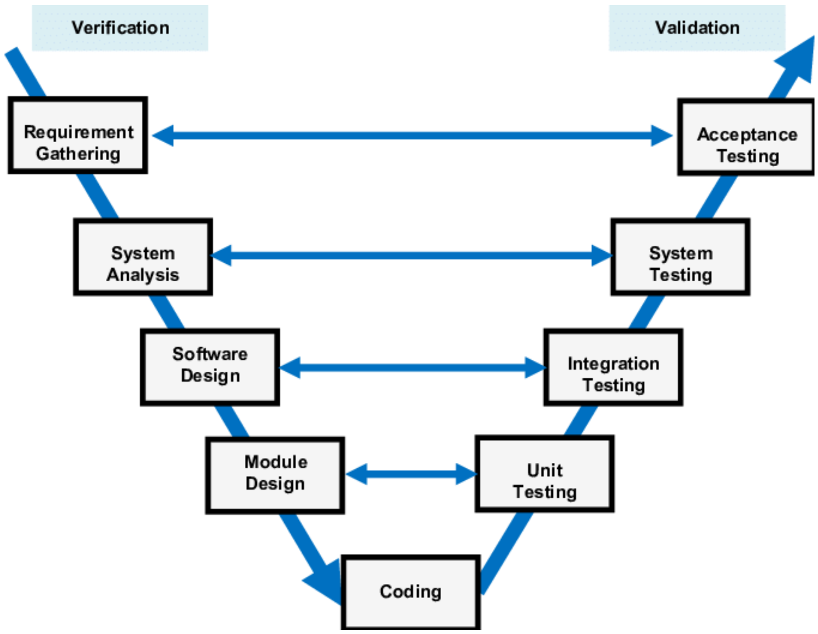


Figure 27. V model

Generally speaking the waterfall model isn't used much anymore due to the reality that software specifications change on a near daily basis.

Recall: aUToronto Spring 2022 integration hell

23.1.1 Agile

Agile is a project management approach which, in most general terms, seeks to respond to change and unpredictability using incremental, iterative work (sprints). This allows for a balance between the need for predictability and the need for flexibility. Some agile methods include:

- Extreme programming: really really fast iteration (think days)
- Scrum: 2-4 week sprints with standups and backlogs; sticky notes for tasks, etc. Think kanban boards. Daily scrum meetings to unblock ASAP. Development lifecycle is therefore a series of sprints.
- On-site customer; frequent interaction with end users to figure out what exactly they need.

Subsection 23.2

I dropped this course

I decided to drop this course because the courseload was a little too much to handle between EngSci ECE, clubs, design teams, work, and trying to have a life.

PART

VIII

ESC301: Seminar

Section 24

Preliminary

Subsection 24.1

Seminar 1