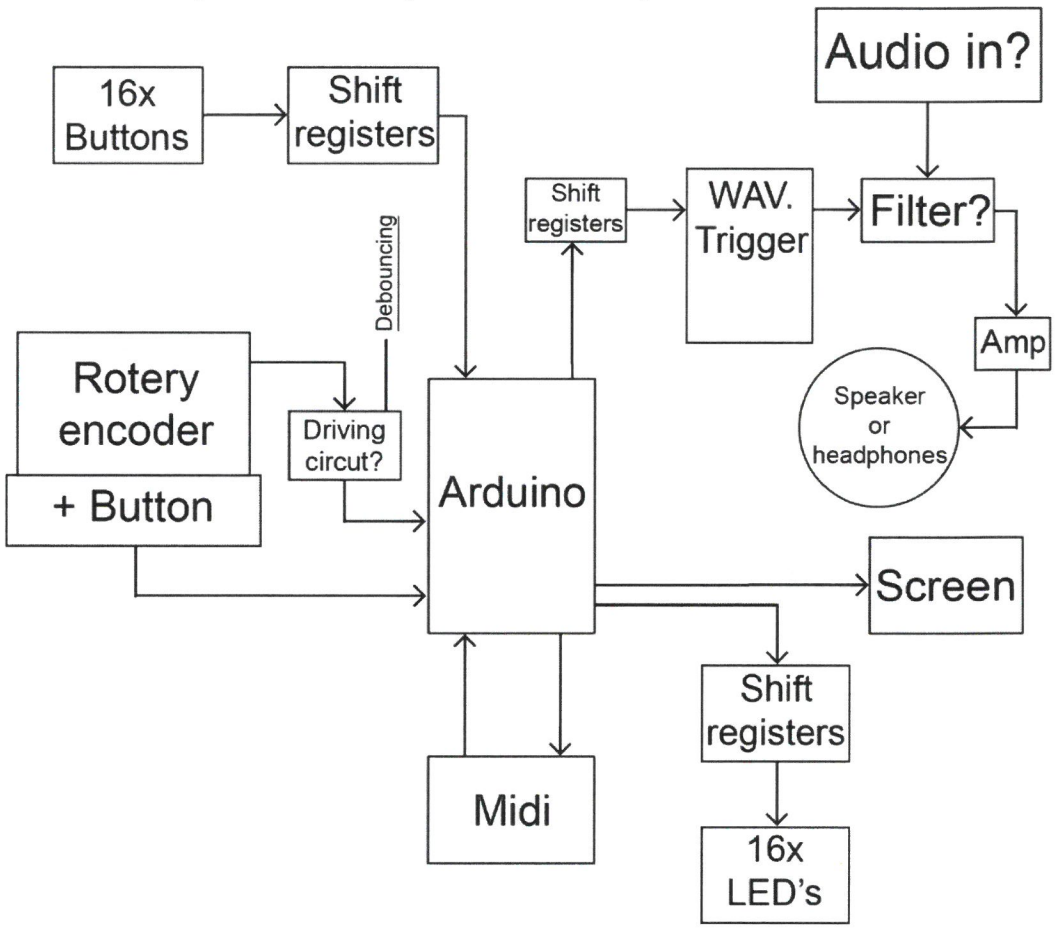
Report for 3.47: Demonstrate understanding of complex concepts used in the design and construction of electronic environments

Specifications:

My design for a 16-step drum sequencer will help explain the hardware and software concepts I have chosen. It will be a standard, variable tempo, 8-voice sequencer. There is an LED and a button for each step of the sequencer, all of which are controlled using bit-shifters. There is a Nokia 5110 backlit LCD screen for displaying information, a rotary encoder for traversing the devices menu with a 3.5mm mono audio jack for the sound. An Arduino Uno R3 (ATmega328P) microcontroller will be programmed to operate the sequencer.

Circuit Schematic

Here is a block diagram of the sequencer control system.



My report demonstrates understanding of five software and hardware concepts respectively in the context of a sequencer:

Complex hardware concepts:

Microcontroller, Bit-shifter, Schmitt Inverter, Multiple Sensors, LCD Display

Complex software concepts:
Structuring programmes logically, Counters, Interrupts, Flags (state variables), Serial data transmission

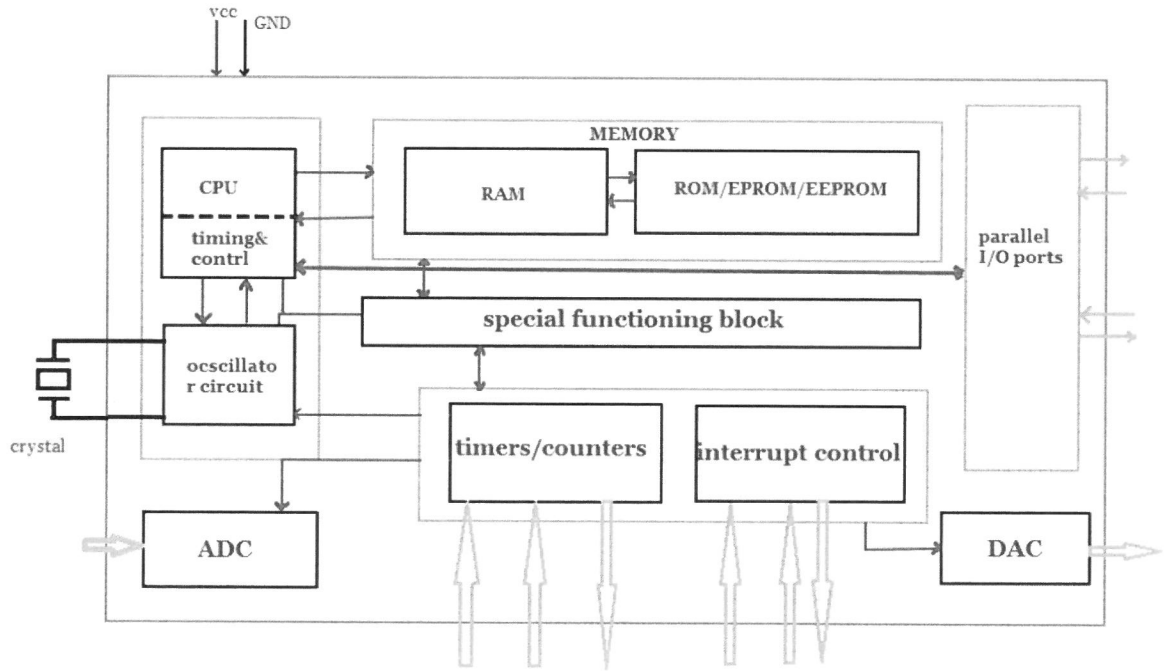
Complex hardware concepts

1. **Microcontroller**

Concept: A microcontroller is a small computer with the footprint of a single integrated circuit that has multiple inputs and outputs with both processing and memory capacity.

Microcontrollers are sometimes referred to as 'computer-on-a-chip', this is because it has its own memory, is capable of processing instructions (code) and data, and has dedicated pins for inputting and outputting data.

A microcontroller contains both ROM and RAM as well as input and output ports. All microprocessors execute programmed instructions in an ordered manner, the execution speed is reliant on the microcontrollers internal oscillator, also known as the clock or crystal (the ATmega328P has an 8MHz oscillator).



A microcontroller is necessary in my sequencer context because tasks such as sensory data, like a button press or encoder turn, and outputting information to the sound chip, screen and LEDs would not be accomplishable without one. Nearly all microcontrollers have enough input and output pins and have support for advanced processing features such as interrupts and ADC (analogue to digital conversion) that are needed for the sequencer. This makes the choice of microcontroller a broad one.

Microcontrollers are classified in many different families, such as: PICAXE, Arduino, Atmel, ATmega and ARM to name a few. Each family, and each individual chip within that family, has different capabilities. These include different memory capacity, amount of I/O (inputs/outputs) and operating voltages. These factors, along with the cost, availability and programming language, will affect which microprocessor is chosen to fit a purpose.

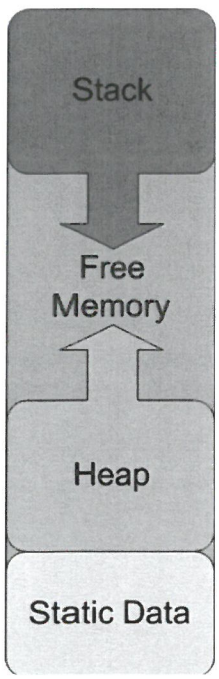
For the purpose of this context, I chose an Arduino Uno R3 (which uses an ATmega328P as a microprocessor) because I am familiar with the programming language (a slightly modified version of C++), they are inexpensive and there is an almost unlimited amount of help and support from both my teacher, and the internet. On top of this, I have around three years experience with these microcontrollers.

The total memory of a microcontroller is divided into two parts, RAM (random-access memory) and ROM (read-only memory). The RAM capacity is generally quite small for most microcontrollers, it is normally about 256 bytes and is used for storing things like values of variables. ROM can be made up of both read-only ROM (this is where information that is necessary for the chip to run is stored called a bootloader, which is similar to an operating system) as well as rewritable ROM (where program code is stored). In both cases, the ROM is non-volatile (remains even after loss of power). The ROM can be made up of EEPROM or Flash memory.

There are three types of memory in an Arduino:

- **Flash Memory**: Used to store program images and any initialized data.
- **SRAM**: Static random-access memory, made up of three parts: Static Data, Heap and Stack. Static data is reserved space for all the global and static variables from the program in flash memory. The heap is for dynamically allocated data items and the stack is for maintaining a record of interrupts and function calls.
- **EEPROM**: Can be read or written to by a program executing from the flash memory. It can only be read byte-by-byte, so it is quite slow and rarely used. However, it is also non-volatile and can therefore be used to store information such as important variable values that may be need when the chip is turned off.

This diagram shows how the different types of memory interact with each other. Static data sits in the bottom of the free memory and is not changed again unless a new program runs. The heap grows from the top of the static data upwards as data items are allocated (such as an analogue value being read). The stack grows from the top of memory down towards the heap. Every interrupt, function call and local variable allocation causes the stack to grow. Returning from an interrupt or function call will reclaim all stack space used by that interrupt or function.

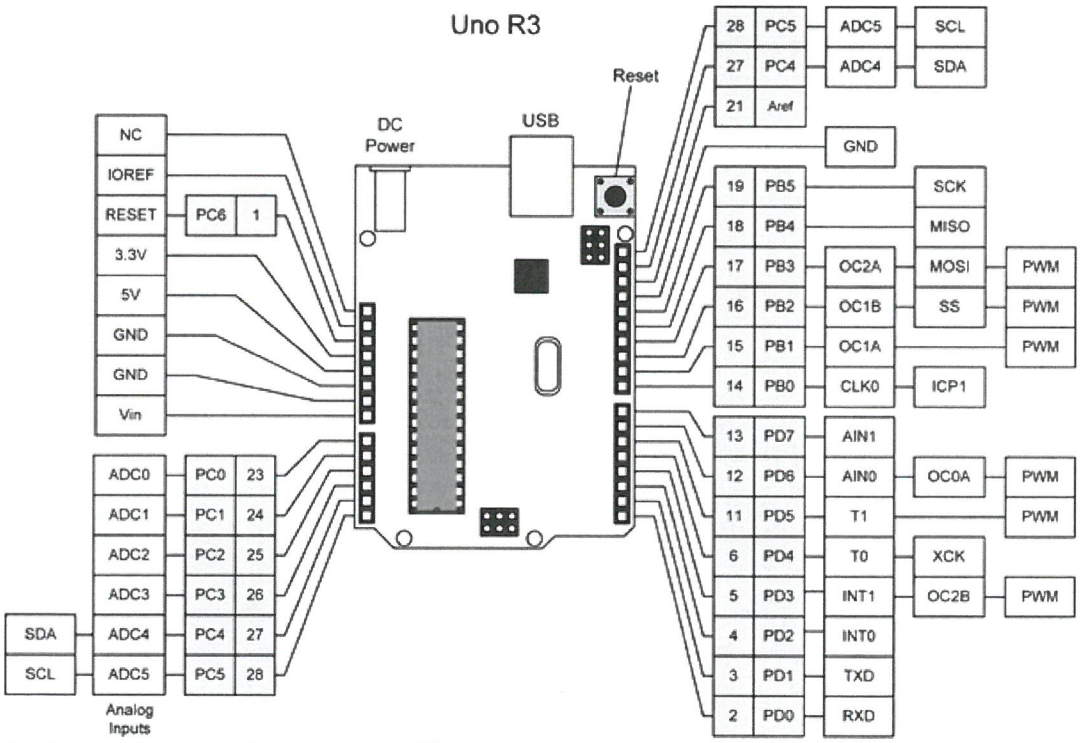


If the stack and heap start overlapping, memory problems will occur. When this happens, one or both memory sectors will be corrupted.

In some cases, this results in an immediate crash, while in others the effects may go unnoticed for some time.

The Arduino IDE gives the user a warning when uploading a program if the memory limit is being approached:

```
Low memory available, stability problems may occur.
```



The Arduino Uno R3 (ATmega328P) was chosen as a microcontroller suitable for the sequencer as is relatively cheap, has plenty of memory capacity (32K) and the exact amount of I/O needed for the other subsystems. This microcontroller also supports the advanced software techniques (such as interrupts and flags) that are needed for the sequencer to function.

2. Bit-shifter

Concept: A bit-shifter is a complex subsystem that can store 8 bits as HIGH (1) and LOW (0) values. Depending on the type of bit-shifter, these values can either be brought into a microcontroller as inputs or used as outputs using only a few pins.

A bit-shifter is a complex hardware subsystem, they consist of flip-flops daisy-chained with one another that share the same clock. These store either a '0' or a '1'. Standard bit-shifter ICs have a total of 8 flip-flops meaning that one byte (8 bits or 8 1s and 0s) of data can be transferred (or stored) at a time.

The SN74HC595N is a widely-used example of a bit-shifter integrated circuit. It is a simple IC that takes the serial input from a microcontroller and converts it into a

parallel output. This specific IC contains 8 D-type data latches (flip-flops), has a wide operating range (from 2 to 6 volts) and has a low power consumption.

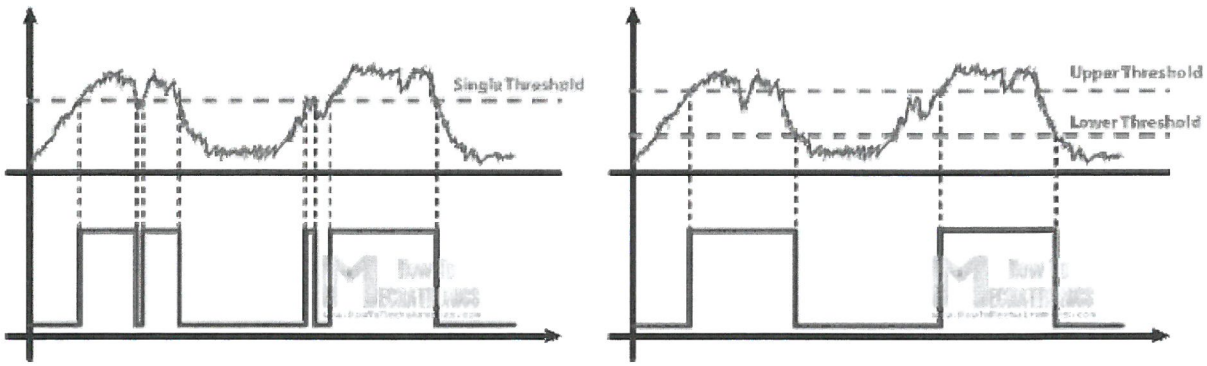
The bit-shifter is used in the sequencer context to allow more efficient control over large amounts of inputs and outputs. For example, this means that 16 LEDs can be controlled using only three pins of an Arduino. You can also daisy-chain them together for an additional 8 outputs without sacrificing any other Arduino pins. This was important information to consider when choosing a microcontroller as more control pins would be needed if a bit-shifter was not used.

The way bit shifters work is by controlling the three pins (clock, latch and data) through code. First off, the latch pin needs to be brought low, this tells the chip that it should be listening for information. In order to shift out, the data is sent through the bit-shifters data line (as either a bool (true or false), high - low, 1 or 0 etc.), the clock line is then pulsed to inform the chip that the last piece of data received should be shifted along. This process repeats until all the data is now stored in the bit shifter, then the latch is pulled high again which tells the chip to stop listening for information.

3. Schmitt Inverter

Concept: A Schmitt inverter is a complex hardware subsystem that is used for beat timing in the context of my sequencer.

A Schmitt inverter (or Hex Inverting Schmitt Trigger) is a comparator circuit that produces a precise square wave from a noisy input when paired with two resistors. The two resistors set the upper and lower threshold of the Schmitt trigger, in the context of my sequencer, I used one fixed resistor and a potentiometer (variable resistor) to control the threshold. This means I can change the speed at which the square wave pulses (referred to as the duty-cycle) by rotating the potentiometer.



The benefit of using a Schmitt inverter is that it helps produce a clean, high – low signal that is easy for a microcontroller to process. I am using the output pulse from the Schmitt inverter gate as the trigger from an interrupt function in my Arduino program.

A Schmitt inverter can also be used to debounce a button. Debouncing a button is important because when a button is pressed, the bouncing generates noise and an

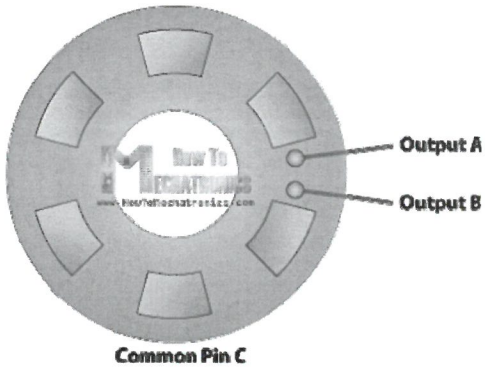
unstable signal. Debouncing eliminates this noise and ensures that the microcontroller only receives the initial press of the button. A simple circuit consisting of a capacitor, a pull-up resistor and a Schmitt inverter debounces a button by creating a low-pass filter (using the resistor and capacitor). The filter reduces noise from the button press before it reaches the Schmitt inverter, once the signal reaches the Schmitt inverter, the comparator waits until the upper threshold is met before pulsing the output. That output is then interpreted by the microcontroller as a single button press with no noise.

4. Multiple Sensors

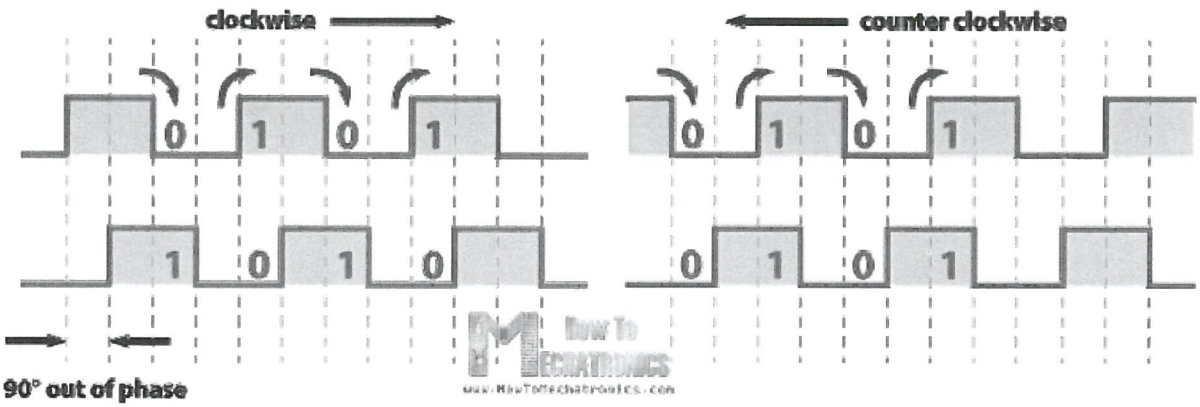
Concept: Inputs from different sensors that can be detected by a microcontroller and applied to an appropriate action. Sensors are components (such as buttons, rotary encoders, LDRs, accelerometers etc.) with their own supporting subsystems. These subsystems include components such as resistors and capacitors arranged in a layout that suits the sensor component. A sensor and its supporting subsystem must be able to output its required values over a range of conditions that the sensor is likely to encounter in its applied context.

The rotary encoder in my sequencer works as an input that tells the microcontroller how it should navigate the software's menu. Incremental rotary encoders (like the one I am using) work by pulsing two output pins high and low, the combination of output values dictate which direction the encoder is turning. The diagrams on the following page help to explain this statement.

The diagram to the right shows how the pins of a rotary encoder are arranged, with a supply voltage going to 'Common Pin C', a square wave is pulsed through both Output A and Output B. Due to the placement of the outputs, the two square waves are out of phase by 90 degrees. We can use this information to understand what direction the rotary encoder is turned



The diagrams below show how the rotational direction of the rotary encoder can be found using the two output pins.

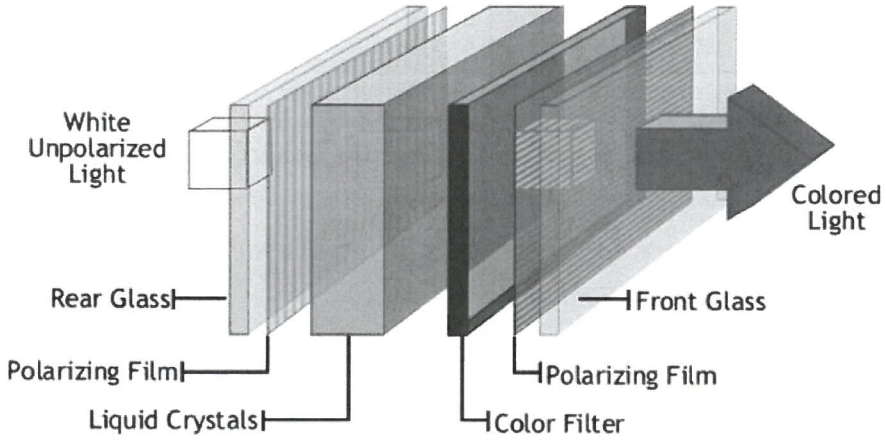


The software can be set up to recognise this information and respond accordingly.

5. LCD (Liquid-Crystal Display)

Concept: An LCD is a flat-panel display that uses the light modulating properties of liquid crystals to display letters, numbers or pixels.

Most liquid-crystal displays (LCDs) are made up of an array of small square 'cells' or pixels that contain light-sensitive molecules sandwiched between two transparent electrodes. When a voltage passes through the electrodes (generally controlled via a transistor switch), the pixel goes dark, then once the voltage is removed, the pixel disappears again. This is due to the liquid-crystal molecules twisting and untwisting when stimulated by the electrical charge from the electrodes. This twisting is what causes the pixel to become visible, when the liquid-crystal molecules are twisted (unpowered), it causes the incident light passing through the rear polarizing film to be rotated to the same angle as the front polarizing film. This means that light can pass through both films without any resistance, which in the case of a monochrome LCD (like the one I am using), the pixel will appear blank and the same colour as the rest of the screen. When a voltage is applied to the electrodes, the molecules almost completely untwist. This results in the incident polarized light passing through that pixel and hitting the front polarizing film. Since the two polarizing films are orientated perpendicular to one another, the light is blocked from passing through and the pixel appears black.



The LCD is used in my sequencer context to display useful information to the user such as Beats-per-minute and what instrument is selected. This information could also be upgraded to display other information such as a metronome.

Complex software concepts

1. **Structuring programmes logically**

Concept: Laying out a software programme so that it can be easily read, understood and improved upon.

There are many ways to improve the structuring of a programme such as using symbols, labels, indentation, annotation and white-space. These techniques improve the format and clarity of the programme making it easier to come back to if need be.

Whitespace and Indentation
Whitespace makes a programme easier to read and follow. Using white-space is almost like organising code into folders, tabbing in a line inside a function makes it clear where it belongs and where following lines should be placed. This makes it faster to find and write code in the programme.

Comments
Comments are areas in a programmes code that are ignored and not compiled. These areas are used by programmers to explain what is happening throughout their code so that they can come back to the code later (when they may have forgotten how the code works), or share the code with another person. Comments are a good way of documenting the program from within the code meaning that extra documentation is not needed to understand and make changes to the code. However, this does clutter up the file and make the code feel quite claustrophobic. Comments do not affect the functional side of the code as they are just ignored sections that are skipped when the programme runs.

Variables
Relevant variables are declared together, this means that finding related variables takes less time and debugging is an overall faster process. A technique called "Camel-case" can be used when declaring variables, functions and classes. The way it works is by having all words that make up an object as a single string with no spaces, except every word starts with a capital letter, excluding the first word.

For example, when declaring a variable, rather than calling it 'three word var' or 'threewordvar', you would call it 'threeWordVar'. Notice how it is much easier to read the name when camel-case is used because you can clearly see where each word starts.

Functions
Functions are a piece of code that is usually used multiple times throughout a programme. A function stores a piece of code and allows for it to be called upon using the functions name rather than having the same piece of code in multiple places throughout a program. This makes a programmes line count smaller and makes debugging much easier because the code only needs to be corrected in one place rather than in the multiple places it would be without functions.

Classes
A class is a collection of functions and variables that are all kept together in one place. These functions and variables can be public, meaning that they can be accessed and used by external code, or private, meaning they can only be accessed from within the classes code. Each class has a special function known as a constructor, which is used to create an instance of the class. The constructor has the same name as the class, and no return type. A classes constructor function is paired with a destructor function, that rather than creating an instance of the class. Is used to remove the instance of the class.

Within the class, there are defined functions that perform certain tasks, the same way regular functions in the above section do. The benefit of using a class however is that large, complex functions (such as the ones for driving an LCD), will be stored in the class and not in the main program, making the code easier to read. Classes are also a good way of including code from other developers because they do not always require a fully written program to run, rather, they are parts of a program.

2. Counters

Concept: Variables that increase (or decrease) when certain events happen. Usually when they reach a certain number an event occurs and the counter is reset.

A common use for counters is looping through an array of data whilst also performing a task for each item in that array. For instance, if the input pins from five buttons are declared in an array, a 'for' loop can be made with an int variable (integer) inside that is equal to zero. After the int is declared, the parameter for the following step is assigned, in the example it is: while the int is less than 4 (i<4). While

that statement it TRUE, the next statement, i++ (increase 'i' by 1), will continue to happen until the pervious statement (i<4) is FALSE. During this time, any code inside the for loop will also run. On top of this, the counter variable can be used inside the loop as well (in place of the array item identifier in the example below).

This example uses a 'for' loop with a counter to set the pinMode of the buttonPins as INPUTS. For each time the loop runs, starting with 'i' at 0 (therefor, starting at position 0 in the array) the pin mode is set as an input for each corresponding pin in the array.

```
const int buttonPins[] = {1, 2, 3, 4, 5};

void setup()
{
    for (int i=0; i<4; i++)
    {
        pinMode(buttonPins[i], INPUT);
    }
}
```

3. Interrupts

Concept: Interrupts are functions that are automatically called when something happens on specific pins.

Interrupts are useful for making things happen automatically in microcontroller programs and can help solve timing problems that normal functions introduce. In the context of my sequencer, I am using an interrupt function whenever the microcontroller receives a parameter change from the Schmitt inverter timing circuit. This allows me to accurately play sounds directly when a beat occurs. This is important because timing is crucial when making musical rhythms.

4. Flags (state variables)

Concept: A variable that has two (flag) or more (state variable) possible values. The program will carry out different functions based on the state of the flag/state variable.

In the context of my sequencer, flags are used to tell whether a sound should be played on a certain beat or not. I have 5 arrays that store 16 different flags each that correspond to a single beat (as it is a 16-beat sequencer). A function loops through each array and depending on whether the flag is a 1 (TRUE, play sound), or a 0 (FALSE, don't play sound), the signal sent to the WAV Trigger will either tell it to play a sound or not.

```
int kick[] = {1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0};
int snare[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
int hat[] = {0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1};
int rim[] = {0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1};
int symbol[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
```

This is of course not the only use for flags, but it is the best example of their use in the context of my sequencer.

5. Serial data transmission

Concept: Sending data sequentially along a wire or wirelessly as a data stream.

Serial data transmission is the sending of data bit-by-bit through a single wire. It is more efficient than parallel data transmission for long distances, but much slower. However, clock speed can be increased to compensated for the slow transmission rate. This method of data transmission also requires less pins on the microprocessor.

Serial data transmission is used in the context of my sequencer when the microcontroller is communicating with the bit shifters. The microcontroller sends and receives serial data from the bit-shifters (both shifting in and shifting out), and the bit-shifter ICs handle the conversion from serial data to parallel data and vice versa. This is an important process as I would have had to use a different model Arduino if the bit-shifters were not capable of converting the data due to the number of extra pins I would need.

Bibliography:

- https://www.arduino.cc/en/Hacking/LibraryTutorial
- http://www.electronics-tutorials.ws/sequential/seq_5.html
- https://learn.adafruit.com/memories-of-an-arduino/arduino-memories
- https://www.safaribooksonline.com/library/view/arduino-a-technical/9781491934319/ch04.html
- http://www.firstpost.com/tech/news-analysis/five-areas-where-the-plasma-tv-beats-the-pants-off-your-lcdled-tv-3625283.html
- http://www.circuitstoday.com/basics-of-microcontrollers
- http://howtomechatronics.com/how-it-works/electrical-engineering/schmitt-trigger/
- http://howtomechatronics.com/tutorials/arduino/rotary-encoder-works-use-arduino/