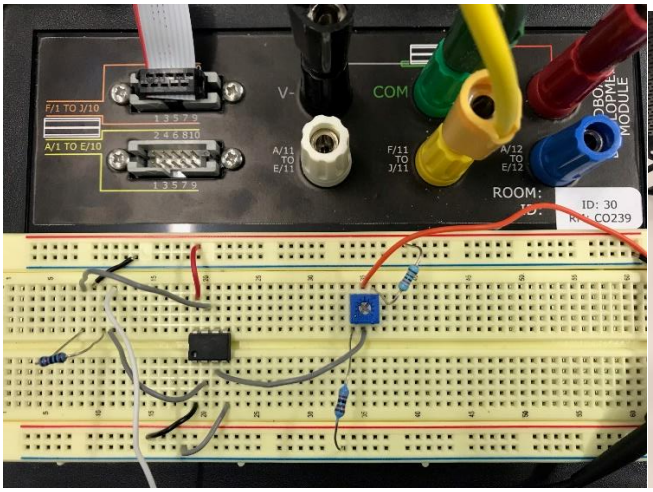
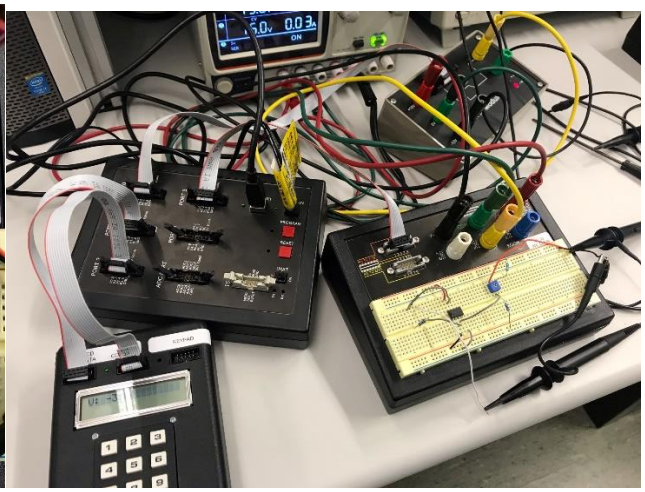
EEEN202, 2021, Microprocessor design exercise

Digital Voltmeter

Project Report: DIY ADC



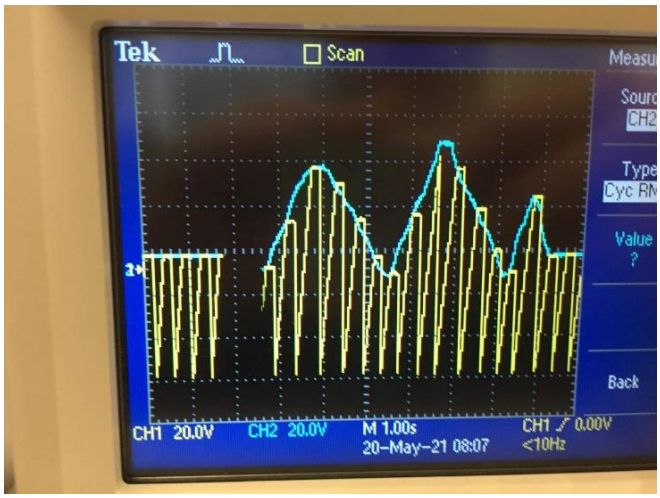


Performance of Staircase Converter vs Successive approximation

Staircase Converter

```
19  // Stair case converter code
20  unsigned char staircaseConverter(){
21      unsigned char test_val;
22      for(test_val=1; test_val<256; test_val++){
23          PORT_DAC = test_val; //set the value of the port
24          delay(100); // wait for the comparator to stabilise
25          if(PIN_CMP){ //check if the comparator is high
26              return test_val;  //return the value if the comparator is
27          }
28      }
29      return 0;
30  }
31
```

A staircase converter uses a binary up counter to sequentially increase the output of the DAC until the comparator until $V_{DAC} > V_{in}$. Upon that state, a signal indicating the "End of Conversion" is sent, and the counter is stopped – the binary value of the counter represents the digital to analogue conversion. On the next conversion, the counter is reset to zero and the process begins anew.

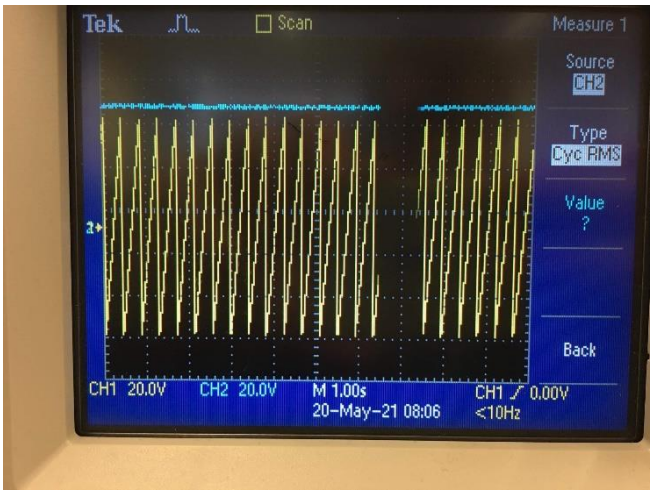


The C code snippet works in the same fashion, though the binary up counter is just a for loop. PIN_CMP is the input from the Op-amp comparator; when it is HI, V_{DAC} is bigger than V_{in} and the loop stops and returns the converted voltage. This creates, as the name implies, a staircase-like pattern of conversion steps when viewed on the oscilloscope

Some samples of conversion being down

As you can see, conversion is not perfect – though this is mainly the fault of the resolution being two low. With max of $\pm 5\ V$, 8 bits give a step resolution of $R = \frac{2 \times 5}{2^8 - 1} = 0.039$. So DAC has the tendency to overshoot the input if it is not a multiple of the step resolution.

The main issue with the staircase converter is how inefficient and slow it is. Because it always starts from zero after a new conversion, conversion times can vary step to step. The iterative checking means that conversion time is also heavily dependent on the incoming signal size - if the signal is at max for a period, it goes through the entire scale multiple times, as shown on the right:



This is very slow and means that if the signal is high frequency, the converted signal will be full of errors as the ADC cannot possibly keep up with the input waveform. To improve the ADC, the staircase converter can be turned into a tracking ADC – a staircase converter which starts the next conversion from the previous value, potentially decreasing the time for future conversions – but this is only a slight improvement at best.

Successive approximation

```
31
32  unsigned char succ(){
33      unsigned char test_val = 0;
34      unsigned char bit_val;
35      PORT_DAC=0xFF; //for scope trigger analysis
36      delay(100);
37      for (bit_val=128; bit_val>=1; bit_val = bit_val = bit_val/2){ //it
38          test_val+=bit_val; //add value of current bit to test_val
39          PORT_DAC=test_val; //set value of port
40          delay(100); //wait for comparator
41          if (PIN_CMP){test_val-=bit_val;} //if comparator is too high,
42      }
43      return test_val;
44  }
45
```

```
45
46  unsigned char bitwiseSucc(){
47      unsigned char test_val = 0;
48      int bit_val;
49      for (bit_val = 7; bit_val >= 0; bit_val--){
50          unsigned char mask = 1;
51          mask = mask << bit_val;
52          mask = mask | test_val;
53          PORT_DAC = mask;
54          delay(100);
55          if (!PIN_CMP){
56              test_val = test_val | mask;
57          }
58      }
59
60      return test_val;
61  }
62
```

Successive approximation is a massive improvement over the staircase converter ADC. In essence, it is an application of a binary search algorithm – an algorithm which searches for an item in an array by comparing the item at the middle and, if the item greater, repeats the procedure in the top half of the array and vice versa for the lower half until the item is found. Successive approximation differs slightly in that the time in each conversion is fixed, whereas binary search has a best case of $O(1)$ (when the middle value is the value we want) and an average performance of $O(\log n)$.

The general procedure for successive approximation is as follows:

1. Clear bits
2. Start at MSB
3. Set bit to 1
    - Is $V_{DAC} > V_{in}$?
        - If yes, reset bit to 0, continue
        - If not, continue
    - Have all the bits been checked?
        - If not, go to next lower bit and start at 3 again
        - If yes, then conversion is complete

So, conversion time is fixed regardless of whether the next sample is much higher than the previous and vice versa – a full conversion would take $n \times t$, or the number of bits times it takes to check each bit.

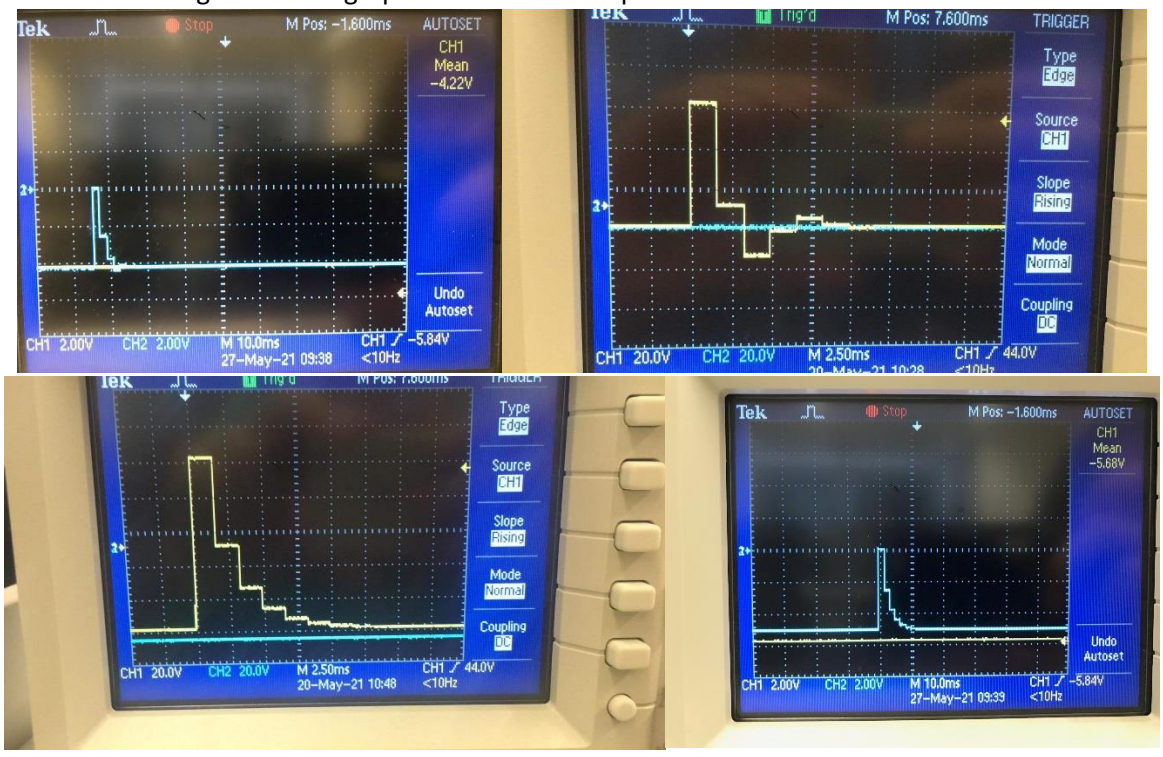
In terms of the C code, there are two different C implementations of the procedure: a manual arithmetic method and a bitwise operation method. The reason for two is that during the process of coding, a suggestion was made to use bit shifting to write the program more efficiently – the result is the same function, but perhaps neater:

```
31 Manual arithemetic method: sequentially adds each bit in decimal form
32  unsigned char succ(){
33      unsigned char test_val = 0;          e.g 128 + 64 + 32 + 16 + 8 + 2 +1
34      unsigned char bit_val;                          (full 8 bit range)
35      PORT_DAC=0xFF; //for scope trigger analysis
36      delay(100);
37      for (bit_val=128; bit_val>=1; bit_val = bit_val = bit_val/2){ //it
38          test_val+=bit_val; //add value of current bit to test_val
39          PORT_DAC=test_val; //set value of port
40          delay(100); //wait for comparator
41          if (PIN_CMP){test_val-=bit_val;} //if comparator is too high,
42      }                  checks if output is higher; if YES, take away
43      return test_val;
44  }
```

```
45 Bitwise operation method: uses bitshifting and logical operations on binary
46   unsigned char bitwiseSucc(){      << : bitshift operator; moves the indexes of
47       unsigned char test_val = 0;       the current bits by specified amount
48       int bit_val;                        e.g 0001100 << 3 == 1100000
49       for (bit_val = 7; bit_val >= 0; bit_val--){
50           unsigned char mask = 1;
51           mask = mask << bit_val;
52           mask = mask | test_val; | = OR logical operations; ORs each individual
53           PORT_DAC = mask;            bit e.g:
54           delay(100);                                  0001
55           if (!PIN_CMP){          0001 | 0010 =>       ||||  =>0011
56               test_val = test_val | mask;              0010
57           }   checks if output is higher; if NOT, OR test_val and mask
58       }
59
60       return test_val;
61   }
62
```

The second method is a more literal translation of the procedure, since it deals directly with bit values, however the main method is really a decimal translation – in fact, a division by 2 after every iteration in the first method is the same as using the right bit shift operator (≫). So, the main difference is really tidiness; in exchange for cleaner code, it becomes a little harder to understand what is going on.

This gives the following waveform graphs in the oscilloscope:



As you can see, the conversion time before the output matches the input is the same regardless of levels – in fact, the conversion happened too fast to be recorded properly sometimes. It still retains the same issue of steps not being fine enough to capture ranges properly so at times the value will not be completely accurate, as shown in the second picture. The current implementation also suffers from the maximums not being captured properly, as shown by the bottom two pictures, where the ADC cannot reach the maximum

negative value. Nevertheless, the successive approximation ADC is a much faster and efficient implementation than the staircase converter due to its smaller fixed conversion time.

Advantages of C language over Assembly language

The advantages C has over Assembly are the same as the question of High-level vs Low-level programming. The key advantage is usability and legibility, where C is much more comprehensible that Assembly. Consider the following code snippets:

Assembly

```
MAIN:   MOV R0,#20
        MOV R1,#0           // Set time value = 0, seconds
        MOV R2,#0           // minutes
        MOV R3,#0           // hours
        ACALL SETDIS        // initialise the display
        MOV TMOD,#0x01      // set timer 0 to mode 1: 16 bit counter
        MOV TH0,#0x3C       // set lower 8 bits to 0x3C
        MOV TL0,#0xB0       // set upper 8 bits to 0xB0
        SETB EA
        SETB ET0            // Enable timer 0 interrupt (your task)
        SETB TR0            // Start timer
```

C

```
void main(){
    double adc_value;
    // int adc_value;
    char output_text[16];
    float RESOLUTION = 2*VMAX / (256 -1);
    initLCD();
    while(1){
        adc_value = -VMAX + /*0.039*/RESOLUTION*bitwiseSucc(); //succ(
        //adc_value = bitwiseSucc();
        //sprintf(output_text,"V: %d", adc_value);
        sprintf(output_text,"V: %0.2f", adc_value);
        writeLineLCD(output_text);
        delay(10000);
        clearLCD();
    }
}
```

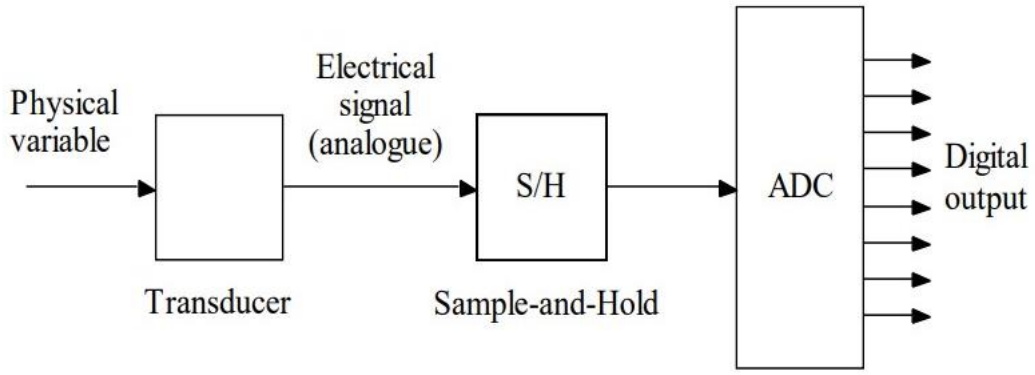
Assembly code is built using mnemonics whereas C has no set structure – you can be as verbose or abstract as you desire. Mnemonics do have the advantage of being fixed and written in a way that gives some contextual clues – ACALL for example is Absolute CALL. But for those starting out in Assembly, the level of abstraction present can be very obtuse and will likely require hours of looking through the microcontroller manual to figure out what each command does. In Assembly, there is a heavy compromise between maintainability and readability vs performance and efficiency – you cannot do both.

C on the other hand is written in plain English (and other languages) – you are in complete control in how legible your code is. This does not completely resolve the maintainability/performance compromise since some programs can still be overly verbose or too obtuse, but code written in our natural lexicon is much more understandable. This ease of programming combined with the wide array of libraries you can use allows for complex programs to be developed faster and maintained easier than a comparably complex Assembly one.

An important part to also consider is that some compilers C allow Assembly code to be written in and certain operations like bit shifting are present. This means that a C program has the potential to be nearly as efficient as an Assembly program despite needing the compiler.

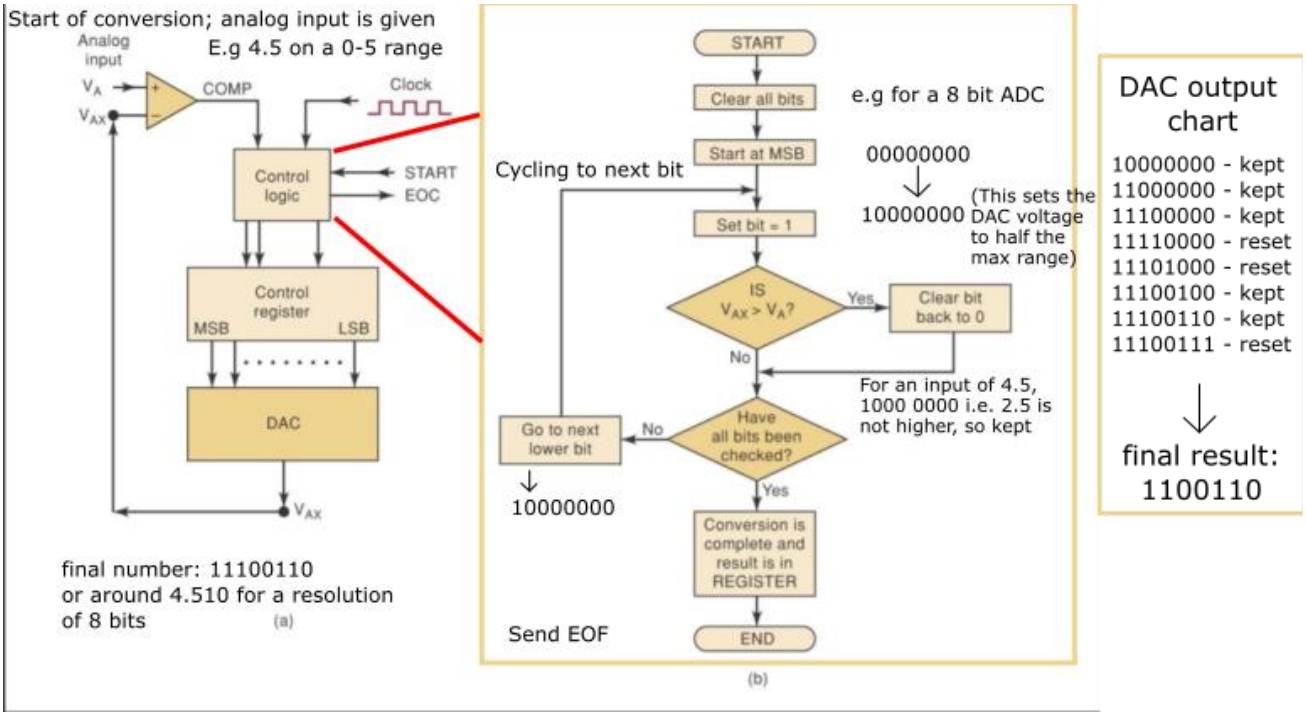
Question section:

1. The diagram below shows a typical data acquisition system where an analogue signal from a transducer is converted to a digital form suitable for subsequent digital signal processing



a. The block labeled ADC contains an 8-bit successive-approximation analogue-to-digital converter.
   i. With the aid of suitable diagrams describe the operation of this device.

The Successive Approximation ADC converts an input signal by a process of binary search - each bit of the ADC digital output is set high and then compared to the analogue input. If the output is higher, then the bit is reset and kept if not. After going through each bit, the output of the ADC should match.



   ii. The ADC is designed to operate over an input voltage range of 0 to 5V. Determine the resolution of the ADC.

$$Resolution\% = \frac{V_I}{2^n - 1} \times 100$$

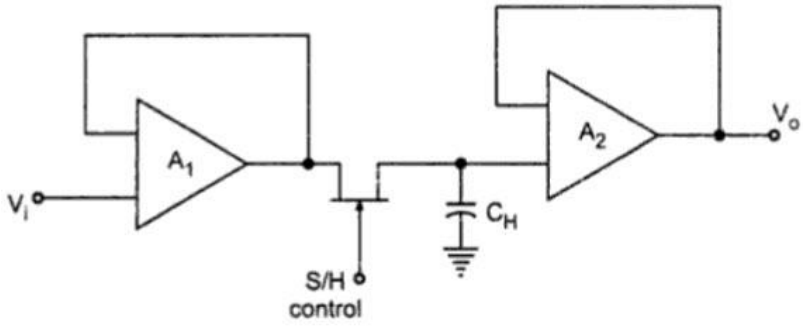
$$= \frac{5}{2^8 - 1} \times 100$$
$$= 1.908\ \%$$

iii. During the conversion process each step takes 2 μs . Calculate the complete conversion time for a full-scale input signal of 5V.

For a successive approximation ADC, conversion time is the number of bits times the time for each step. So for an 8 bit ADC with $2\mu s$ step, each conversion takes $16\mu s$

b. Another of the blocks in the diagram is labelled sample-and hold.

i. Explain the purpose and operation of this device.



The sample-and-hold circuit essentially takes a "snapshot" of the current voltage of a time varying signal, which is useful for a very high frequency input signal, especially when a suitably fast ADC is not available/feasible. A typical circuit can consist of two Op-Amps and a capacitor with a control line for sampling and holding. When the circuit is sample mode, the switch is closed and the capacitor is connected to the buffer amplifier, charging to the output voltage – in hold mode, the switch is disconnected, and the signal voltage is stored in the capacitor, which can be read through the second Op-amp. This enables the signal duration to be increased, allowing the ADC conversion to occur without significant errors from the signal switching too fast.

ii. The sample period is 25μs. Find the highest frequency component that may be present in the signal before aliasing occurs.

Nyquist-Shannon theorem states that the minimum sample rate is greater than 2f to avoid incorrect sampling. So:

$$f_{sampling} = \frac{1}{25 \times 10^{-6}} = 40\ kHz$$
$$f > 2f \quad \therefore \quad \frac{40}{2} = 20\ kHz$$

So, the highest frequency that may be present must be below 20 kHz

2. It is often necessary to acquire data from several analogue input channels. Describe how a simple addition to the above data acquisition system can convert it to a system which can digitise four input channels of data and yet use only one ADC unit. (There will be a corresponding reduction in the sampling rate for each channel).

Several analogue inputs can be digitised together with one ADC by multiplexing the signals. The multiplexer inputs are set to the analogue inputs and a clock with a MOD4 counter, which cycles through each input, sampling the signals sequentially. This however reduces the sampling rate for each channel regardless of how fast the clock can switch the mux.

3. For the following analogue to digital converter case, calculate both the binary and decimal output code.
    i. An input voltage of 3.293 V into a 8-bit ADC with a 0 – 10 V input range.

$$R = \frac{10}{2^8 - 1} = 0.03921\ldots$$
$$\frac{10}{R} = 83.9715 \approx 84$$
$$84 \rightarrow 0101\ 0100$$

4. A 16 bit ADC system with a sample period of 20μs is used to record a song that lasts for 5 minutes. Calculate how much memory is required to store the song.

$$\frac{5 \times 60}{20 \times 10^{-6}} = 1.5 \times 10^6\ 16\ bit\ samples$$
$$1.5 \times 10^6 \times 16 = 24 \times 10^6\ bits\ or\ 3\ MB$$

5. Discuss the difference between SRAM and DRAM.

Both refer to Random Access Memory – volatile memory used for temporary storage – and the difference comes from how the bits are stored, which leads to several consequences for both:

SRAM: Static RAM – bits are stored using flip-flops; data persists so long as power is applied. Usually very fast access times, with static timings; often used for cache

DRAM: Dynamic RAM – bits are stored using capacitors; due to capacitors being imperfect, their voltage drains over time, so bits need to be periodically refreshed while being stored by a DRAM controller. Low power requirement, moderate access times, and very high capacity – used for main internal memory in PCs

6. Discuss the difference between SDR and DDR DRAM

SDR stands for Synchronous DRAM, which as the name implies, is synchronous with the computer's clock with operations executing on the rising edge. This enables data access to operate more efficiently than standard asynchronous DRAM, which are affected by propagation delay. Memory cells are also organised in sections called banks, allowing data access commands on each bank to occur simultaneously, increasing throughput and enabling burst ordering – a process where transfer time is reduced by storing memory words being stored in order so access time is reduced. DDR refers to "Double Data Rate" DRAM, where a memory word is transferred on both clock edges, enabling a lower clock frequency to effectively have the same or higher data throughput.