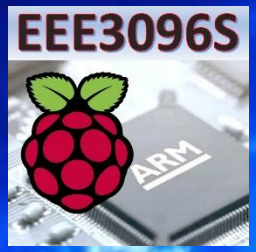
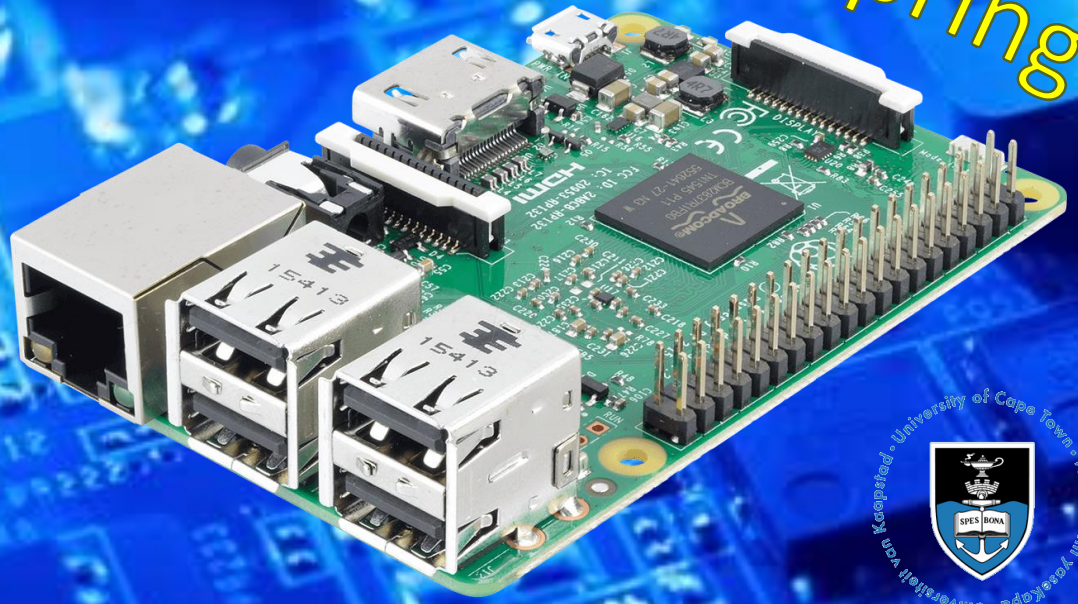


Spring Special



Electrical Engineering
University of Cape Town

P04 - SPI & Interrupts on Raspberry Pi

Some of the content is examinable (e.g. ADC aspects) but specifies about Prac4 and the modules are not.

Dr Simon Winberg

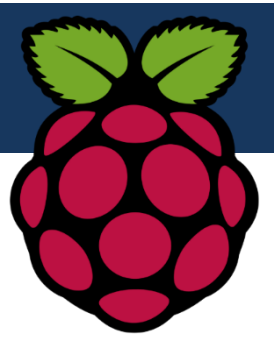
Embedded Systems II

P4

Outline

- Overview of Prac4 aspects
- Onwards to an awesome Prac4 & possible IoT
- Sensing and background on sensors & ADC used for Prac4
- Raspberry Pi spidev library
- RPi.GPIO interrupt facilities
- Quick ADC-related activity

Prac4 - overview



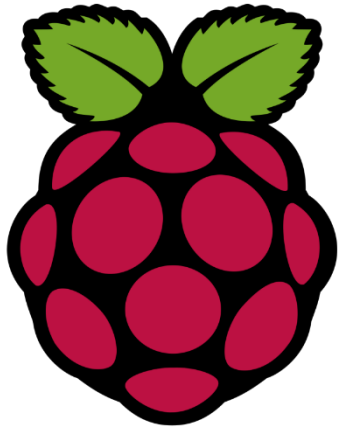
- Prac4 provides an exercise of connecting to an ADC via SPI and of implementing callbacks functions that provide a (sort of) interrupt service routine (ISR).

Why 'sort of' – because there's a whole lot of stuff hidden from you behind the scenes, you're using RPi.GPIO functions to save time.

Perhaps you are done already?! *Yaay!* if so, bit if not it doesn't matter because these slides supplement your understanding of what is going on in the libraries used in Prac4.

Prac4 – adding some more challenge

- If you want to take things further (not for marks)…
  - So the plan is to complete Prac4 just using Python, should be doable in about 2h (?!)
  - But you could try to do be more fancy, try things from first principles e.g. using assembly or using the WiringPi (more challenging that Rpi.GPIO)



Prac04 …

Let’s get started

Prac 4 – an environment sensing system

ADC:

- MCP3008 IC
  10bit ADC – the star of the show)

Sensors:

- MCP9700A (temp sensor)
- LDR (light dependent resistor)
- 1x 1K POT (because we can)

Interface:

- Three switches (stop/start, change sampling speed, display)

Sensing

Design decisions …

Clearly none of these sensors need to be sampled particularly frequently

- Temperature is probably not going to change much over 1s
- The light might change quicker, but we're probably more interested of an average over time (e.g. over an 1h period)
- The POT is just a dummy sensor, meant to actually simulate a 'particulate sensor'* (more particulates in atmosphere the more the resistance). Usually a particular sensor would be read quite quickly ~ every 2ms but we can sample our POT slowly

* A particulate dust sensors typically use a light sensor in a dark cavity to measure the transparency of the air, usually has a little fan to pull in air. They cost around R500.
(You can actually make your own for cheap)

Grove™ optical-based dust sensor

Recording and Displaying

Design decisions … towards an IoT device

Environment monitors usually log the sensor readings. Ideally you would want to send it somewhere via internet (i.e. an IoT device) but in this case you can just record the latest data in variables, or you can implement an array in memory that captures to the last N samples.

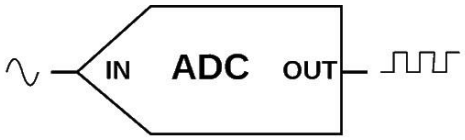
Time	Timer	Pot	Temp	Light
10:17:15	00:00:00	0.5 V	25 C	10%
10:17:20	00:00:05	1.5 V	30 C	68%

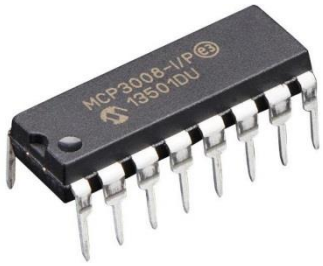
You need to display the latest readings with the display function.

Optional extras if you want to play around further with the RPi:

- This project is in a way a basis for doing more with the RPi, you could extend it to become a **real IoT sensor**, send you arraw of recorded sensor readings over the internet, say every 10 minute, to a server that will log the data into a csv file, and could display averages and other stats.

The MCP3004/3008 SPI Analogue to Digital Convertor



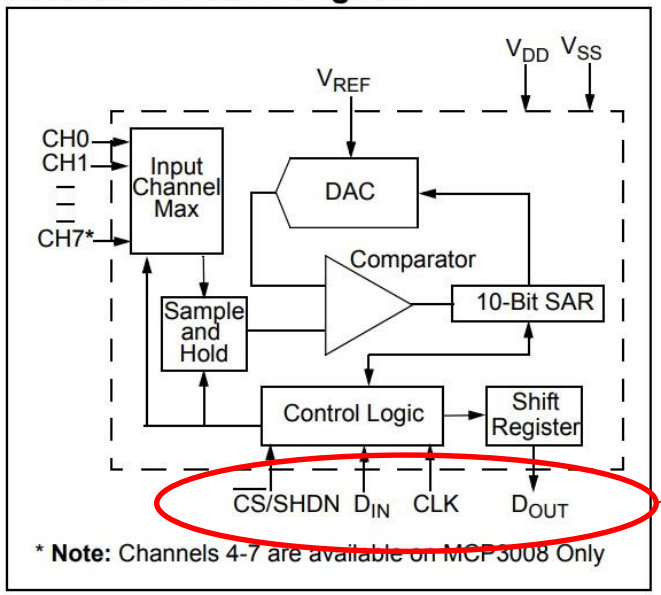


- A small but effective ADC
- Uses SPI serial interface so very few pins needed to get the readings

Note speed limitations

- 200 ksps max. sampling rate at VDD = 5V
- 75 ksps (bit disappointing) max. sampling rate at VDD = 2.7V

Functional Block Diagram



For this application you don't actually need to run it fast at all, you're essentially just going to read it every now and then, won't be anything close to the maximum rate.

This is essentially a low-cost ADC, not a fancy flash ADC

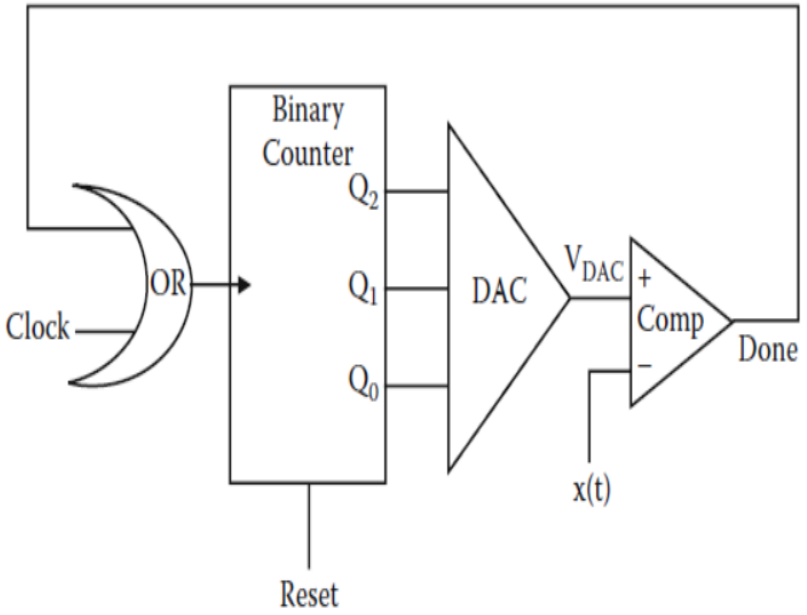
Interface you will be using

Conceptual ADC operation

Simplified operation (MCP3004/3008 is a bit fancier)

Digital-to-Analog Converter Transfer Function

ADC Output Code (Q2, Q1, Q0)	V_{DAC} (volts)
000	–0.75
001	–0.50
010	–0.25
011	0.00
100	0.25
101	0.50
110	0.75
111	1.00



While this is a conceptual example of how a ADC works, it is also quite in line with how a low-cost ADC works, where there is just a counter and a DAC to compare when the voltage corresponding to the value counted up to has reached the reference voltage.

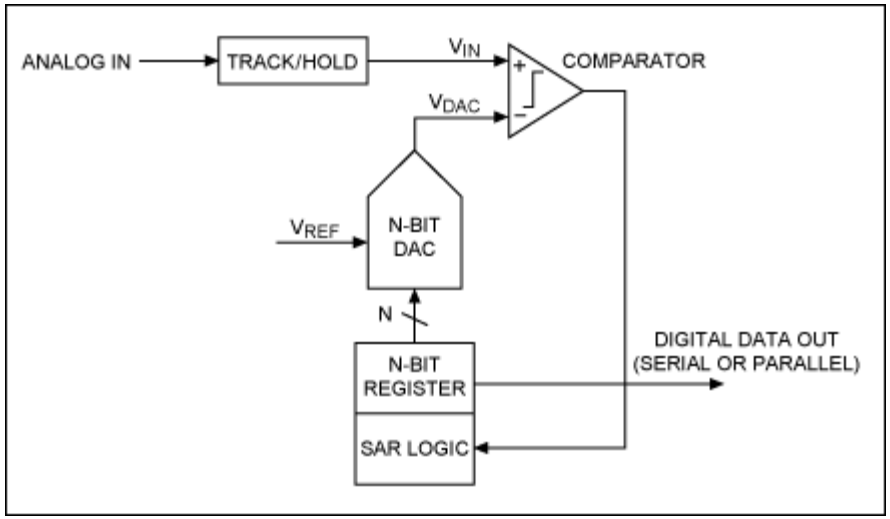
? *PS:* How much have you done about ADCs already? Do you already know this model? Do you know what e.g. ENOB means? How to design a FLASH ADC?

The MCP3004/3008 SPI Analogue to Digital Convertor

In reality the MCP3008 is a SAR-based ADC, which is a compromise between a simply counter-based ADC and a high-speed FLASH, it's performance is somewhere in between, often referred to as 'midrange' ADCs.

Basically it uses a single comparator and a binary search method to locate the reference voltage more quickly (at well as some other cleaver ticks)

PS: If we haven't already covered SAR-ADC in ES 1 we had better to so because these are essentially the most common types of ADC in the market nowadays.

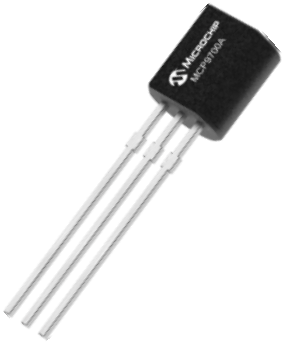


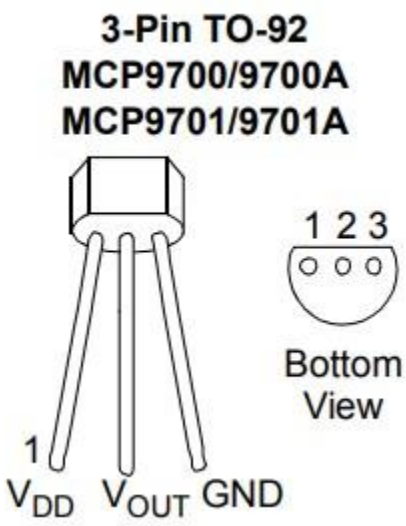
Simplified N-bit SAR ADC architecture.

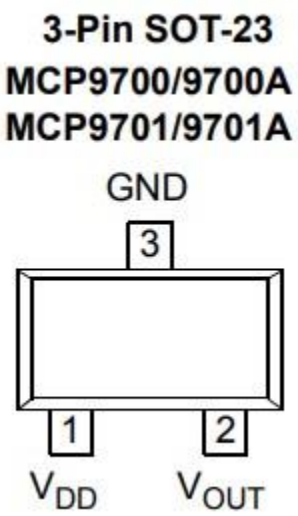
Image adapted from https://www.maximintegrated.com/en/app-notes/index.mvp/id/1080

The MCP9700A Temperature Sensor

- A tiny analogue temperature sensor
- Yes analogue as an excuse for using the ADC ☺







Easy to connect up, e.g. using a breadboard, and sampling V_{OUT}

Make sure you condition the signals to the ADC so that you have the right voltage that the ADC can effectively sense.

Typical Working of a Linear Temperature Sensor

How to measure temperature:
This sensor generate output voltage proportional to temperature.
Given by a formula such as:
$V_{out} = T_C \times T_a + V_0$

where:
V_{out} = Sensor output voltage
T_C = Constant value that increase output voltage in every 1.0°C (e.g. 10mV)
T_a = Ambient temperature
V_0 = Sensor Output Voltage at 0°C (500mV)

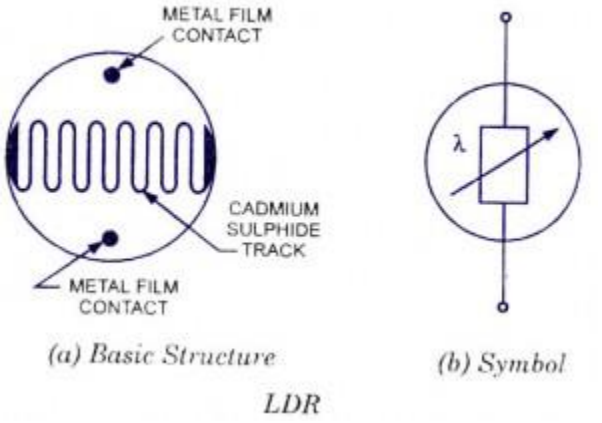
Thus temperature is calculated using the formula:
$T_a = (V_{out} - V_0) / T_C$

You may want to calibrate the system using know temperatures or thermometer, e.g. have two known temperatures and see what readings these correspond to.

Light Dependent Resistor (LDR)

- A wonderfully cheap device
- Just changes resistance depending on brightness of light shining on it

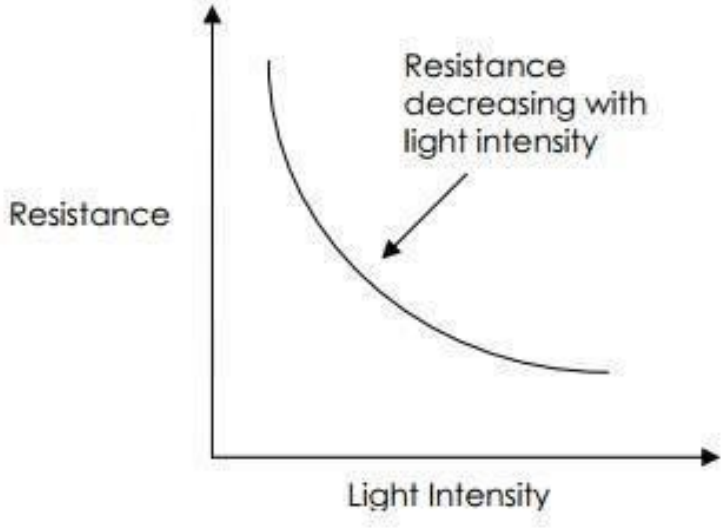
The system works essentially by the characteristic of cadmium sulphide that changes it resistance (slightly) as the light changes.

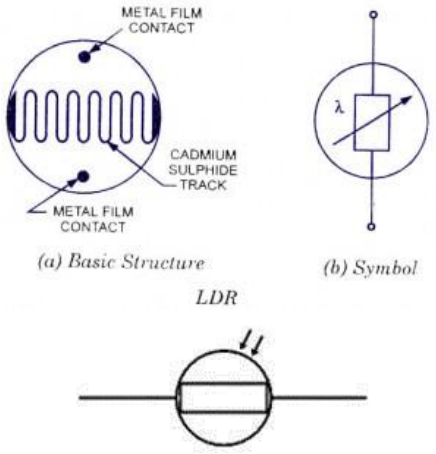


How it works (in brief)… (next slide)

Light Dependent Resistor (LDR)

Operation



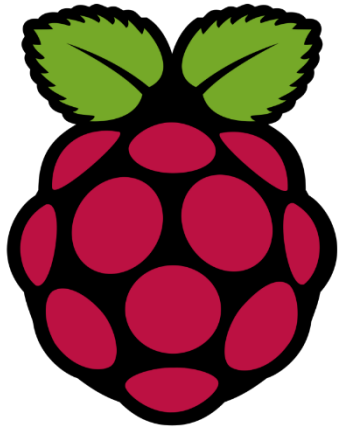


(c) Alternate symbol

The resistance variation in relation to changing light intensity

Most common types of LDR has a resistance that <u>falls</u> with an increase in the light intensity falling on the receptive CdS area. Typically resistances:
Daylight : 5kΩ
Dark : 20MΩ

As the diagram indicates a negative exponential distribution, i.e. it is more sensitive to lower light conditions and after a certain point these is essentially little change. A common design decision is to put a window (or filter) in-front of the LDR to adjust the sensor for the environment it is meant to operate in – often they have such a thing in place suited for normal outdoor or shade operation.



Code Resources for Prac04
SPI and
RPi.GPIO Interrupts Utility

Raspberry Pi spidev library

- The spidev* library is a library for accessing SPI devices through the Linux kernel driver.
- It is somewhat more optimized than RPi.IO because it manages the files used to stream data to/from SPI devices more efficiently
- It is simple, you just need to know which SPI bus and which devices on the bus to use

Example code:

```
import spidev
spi = spidev.SpiDev()
spi.open(bus, device)
to_send = [0x01, 0x02, 0x03]
spi.xfer(to_send)
```

* https://pypi.org/project/spidev/

spidev behind the scenes

Behind the scenes spidev uses various (C-based) precompiled code to make things happen.

The code is pretty much like that shown on the right where a **peripheral register structure** is at the heart of the operation.

Essentially the PIO registers concerned get configured. The ARM Cortex (within the BCM2837) can have some of its pins set to 'GPIO Alternate Function' mode (i.e. AF Mode in datasheet).
This allows the pins to be linked directly to internal clocks. It also links an 8-bit shift register to the input and another to the output. So it can shoot out an 8-bit SPI packet concurrently while the CPU does other stuff. This makes SPI operation incredibly more robust than 'bit-banging' in software.

```
void mySPI_Init(void){

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_SPI1, ENABLE);

    SPI_InitTypeDef SPI_InitTypeDefStruct;

    SPI_InitTypeDefStruct.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
    SPI_InitTypeDefStruct.SPI_Mode = SPI_Mode_Master;
    SPI_InitTypeDefStruct.SPI_DataSize = SPI_DataSize_8b;
    SPI_InitTypeDefStruct.SPI_CPOL = SPI_CPOL_High;
    SPI_InitTypeDefStruct.SPI_CPHA = SPI_CPHA_2Edge;
    SPI_InitTypeDefStruct.SPI_NSS = SPI_NSS_Soft;
    SPI_InitTypeDefStruct.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_2;
    SPI_InitTypeDefStruct.SPI_FirstBit = SPI_FirstBit_MSB;

    SPI_Init(SPI1, &SPI_InitTypeDefStruct);

    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA | RCC_AHB1Periph_GPIOE ,
    ENABLE);

    GPIO_InitTypeDef GPIO_InitTypeDefStruct;

    GPIO_InitTypeDefStruct.GPIO_Pin = GPIO_Pin_5 | GPIO_Pin_7 | GPIO_Pin_6;
    GPIO_InitTypeDefStruct.GPIO_Mode = GPIO_Mode_AF;
    GPIO_InitTypeDefStruct.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitTypeDefStruct.GPIO_OType = GPIO_OType_PP;
    GPIO_InitTypeDefStruct.GPIO_PuPd = GPIO_PuPd_NOPULL;
    GPIO_Init(GPIOA, &GPIO_InitTypeDefStruct);

    GPIO_InitTypeDefStruct.GPIO_Pin = GPIO_Pin_3;
    GPIO_InitTypeDefStruct.GPIO_Mode = GPIO_Mode_OUT;
    GPIO_InitTypeDefStruct.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitTypeDefStruct.GPIO_PuPd = GPIO_PuPd_UP;
    GPIO_InitTypeDefStruct.GPIO_OType = GPIO_OType_PP;
    GPIO_Init(GPIOE, &GPIO_InitTypeDefStruct);

    GPIO_PinAFConfig(GPIOA, GPIO_PinSource5, GPIO_AF_SPI1);
    GPIO_PinAFConfig(GPIOA, GPIO_PinSource6, GPIO_AF_SPI1);
    GPIO_PinAFConfig(GPIOA, GPIO_PinSource7, GPIO_AF_SPI1);

    GPIO_SetBits(GPIOE, GPIO_Pin_3);

    SPI_Cmd(SPI1, ENABLE);
}
```

SPI initialisation on an STM

Threaded Callback Handler on RPi

- The RPi (actually embedded Linux / Raspbain) uses the threaded callback handlers
- For this prac we use the RPi.GPIO interrupt facilities
- It is quite easy to make use of…

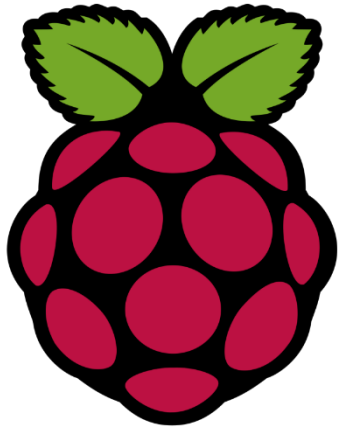
Step 1: define your callback

```
def callback1(channel):
    # put code here

    # channel relates to which
    # io source was responsible,
    # so you can use one callback
    # for multiple interrupts
```

Step 2: link up the callback

```
# When a falling-edge is detected
# callback function will be called

GPIO.add_event_detect(switch_1,
    GPIO.FALLING, # type of signal change
    callback=callback1,
    bouncetime=200) # optional other feature
```

NB: See Interrupts Lecture to see details on what actually a Threaded Callback Handler is



You’re now all set for Prac4 !!

Conceptual ADC Operation

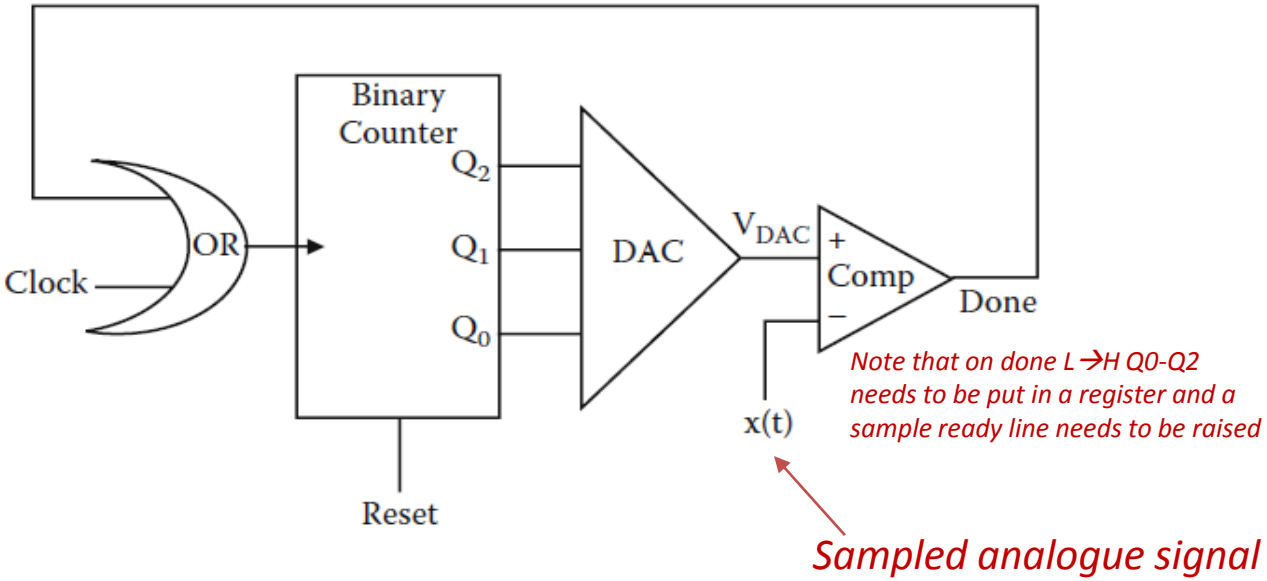


FIGURE 7-1 Conceptual analog-to-digital converter.

Notes about this conceptual representation:

- *This conceptual ADC is used to illustrate the operation of an ADC, which it could be implemented like this it would not be very efficient.*
- *In practice, a sample-and-hold (SAH) circuit prevents the ADC input, a time-varying analogue voltage x(t), from changing during the conversion time.*
- *The counter stops counting when its output, which is converted into a voltage by the DAC (digital-to-analogue converter), equals x(t), indicated by comparator. The counter output is then the ADC output code corresponding to x(t).*

Figures and descriptions adapted from: Martinez, David R., M. Michael Vai, and Robert A. Bond. High performance embedded computing handbook: A systems perspective. CRC Press, 2008.

Quick Activity / Exam Question

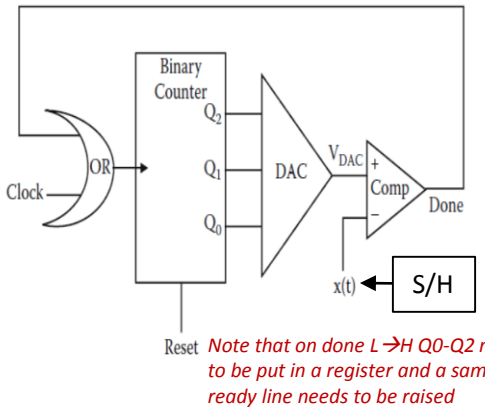
Consider .

A super basic single comparator counter-based ADC that has a 10-bit resolution (this is hypothetical of course). If we wanted it to give a result equally quick as the MCP3008 at 2.7V (ie. running at 75 Ksps sampling rate, i.e. where K = 1024)

Then:

a) What rate must it be clocked at for reading the worst-case readings (e.g. if max. *x(t)* voltage input)?
(i.e that Clock input must run at)

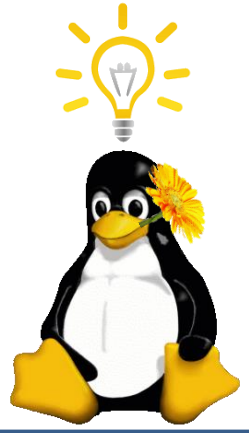
b) If you are using cheap CMOS (fancy CMOS can do a few GHz) that can support a maximum of a 400MHz switching, can this simple counter approach be supported using this technology? (motivate / show working your answer. *



Note that on done L→H Q0-Q2 needs to be put in a register and a sample ready line needs to be raised

3-bit counter ADC

* Don't fall into my trap – this isn't as simple as it looks, you may need to get out and apply that thinking cap.



Quick Activity Answers

Sample solution…

Q:

(a) What rate must ADC be clocked at for reading the worst-case readings?

A:

We know it is max 2^{10} = 1024 values to be cycled through for one reading.

It is running at 75K samples per second.

So we need: 75 x 1024 x 1024 cycles per second

= 75 MHz

Quick Activity Answers

Sample solution…

Q:

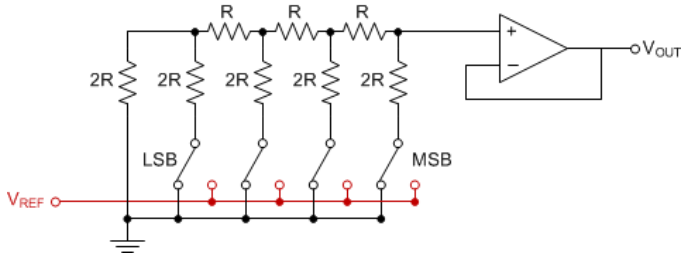
(b) If using a cheap CMOS that support maximum of a 400MHz switching, can this simple counter approach be supported using this technology?

A:

Well from a first impression, we already know that the clock is supposed to be running at 75 MHz, this may look safe enough away from the 400MHz limit. But it's not that far. So we need to think what kind of switching may happen between one clock and then next.

These are the suspects to consider:

- OR gate – Obviously same, just 400MHz
- Compactor – analogue component so probably ~200MHz rate
- DAC – essentially analogue also (FETs and resistors, see illustration right)
- Counter (i.e. a 10-bit ADDER) – Aha, now this isn't so obvious



Conceptual 4-bit DAC

Quick Activity Answers

Sample solution…

Q:

(b) If using cheap CMOS that support maximum of a 400MHz switching, can this simple counter approach be supported using this technology?

A:

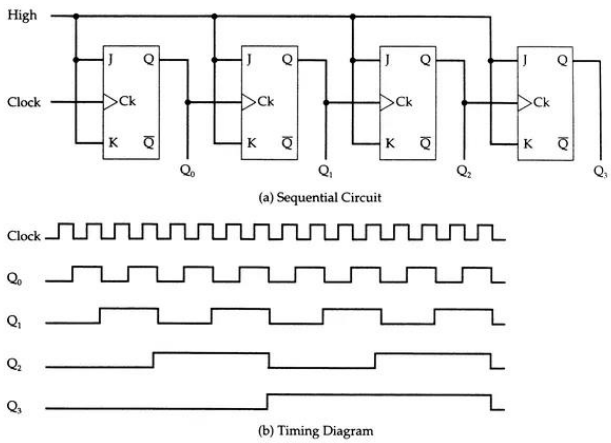
We've decided that we need to investigate the counter further…

As you may know, the simple implementation is the ripple counter. Called ripple counter because it is frustratingly slow, the carries have to ripple from the LSB up to the MSB.

So time to complete (ignoring last carry) would an increment is:

10 x gate delays. 400M gate delays per second → 400M / 10 = 40M is the max speed of the adder. Therefore ~~BOOM~~, too slow for the slow.

But we aren't done yet…!



4-bit ripple counter operation

Quick Activity Answers

Sample solution…

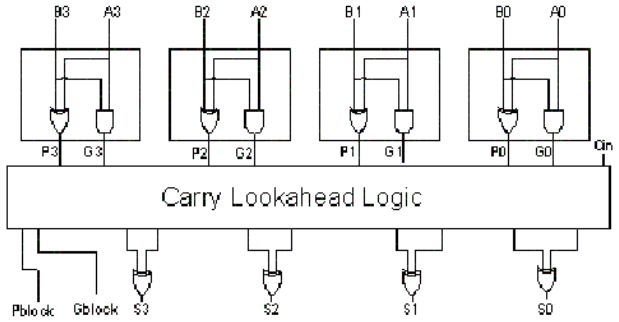
Q:

(b) If using cheap CMOS that support maximum of a 400MHz switching, can this simple counter approach be supported using this technology?

A:

We've decided that we need to investigate the counter further…

But what of more efficient adders, such as (your wonderful friend mention in ES1?) the carry look-ahead adder (CLA)



Conceptual design of CLA

It's actually quite difficult to figure out the gate delay for the CLA, but if one has experience of them it helps. Basically for an N-bit added the worst case can be around N/2 (that doesn't help much!) or about ceil(log2(N)) we know $\log_2 10 = 3.3 \rightarrow 4$ so we get 400M/4 = 100MHz… which is just about right ?! *

NO! Answer is probably no, because 100MHz is basically the best case, sometimes it may take longer, and besides there are other components in the circuit to do stuff after the counter.

You're probably thinking I was intentionally annoying and worked it out to give potentially confounding results (true, maybe so ☺)

The Next Episode…

Lecture P19

Code Resources for Prac04
Details on ADCs (not needed for Pracs) &
Interrupts complexities

References

- https://en.wikipedia.org
- https://en.wikipedia.org/wiki/Carry-lookahead_adder
- Insights on using SPI on the ARM: https://www.voragotech.com/sites/default/files/Vorago%20SPI%20block%20use-%20%20AN%20-%20v1.0.pdf