1 MHz = 1,000,000

1MHz / 8 = 125,000

256 / 125,000 = 0.002048

Analog I/O

- The physical world is a analog
 - On/off switch inputs are insufficient
 - Simple user info outputs include LEDs, NVM, displays
- Inputs
 - Analog input are from sensors, they come in many forms
 - Temperature, pressure, voltage, light, acceleration (2D/3D), magnetic field strength(Hall effect)
 - We get a voltage and need to convert that into something meaningful.
 - We need knowledge of the sensor characteristics.

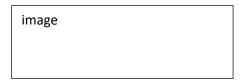


- 2 ways of getting the voltage
 - ADC units (Chapter 26 in manual)
 - A special function unit (s.f.u.) that take an input voltage and puts a value in a register
 - Number of bits gives us resolution
 - We need a reference voltage to define a maximum voltage and thus our step values
 - E.g. 5v AREF & 10-bit register
 - \circ ADC = ($V_{in} * 1024$) / V_{AREF}
 - o 0x0000 => 0v
 - o 0x03FF => 5v
 - o 0x01FF => 2.5v
 - Still need on more conversion to get real value
 - E.g. voltage => degree Celsius (C)
 - Manual 26.8
 - 2) frequency capture
 - Uses a timer
 - S.f.u. (capture unit) writes timer value to a capture register
 - Writes based on seeing a negative or positive edge on an input
 - Can fire an interrupt
 - We can take the time between captures to determine frequency.
 - Time between positive & negative edges gives us the duty cycle

- E.g. RPM
 - o Count edges over a time period
 - X pulse over y ms
 - Get 1 pulse/revolution
 - o Via hall effect
 - Easy RPM calculation
 - Chapter 17.6 in manual
 - More details later (assignment 3 & 4)

Sampling analog inputs

- All analog inputs vary, so when we read has an effect on what we see
 - o E.g. sine wave



- When we read can give us radially different values
- Must sample multiple points and average the values (integrate) to get an accurate value
 - More samples give us more accuracy
 - Problem:
 - Take longer time to get analog values
 - If we sample too fast we will also see bouncing as input oscillates
 - Need more samples to damper the oscillation
- How do we calculate a running average efficiently
 - (1) Easiest way (worst way)
 - Continuously re-calculate by summarizing our values
 - Problem: slow & repeated work
 - (2) second way
 - Get an average with full history
 - Add in next value and do a multiply/divide
 - Problem: it's a full history
 - o Includes all samples (slow)
 - We don't care about all old(history) data
 - (3) sliding window (best way)
 - Store our last N samples
 - With each sample we read, we throw away the oldest history data, add(replace with) the new data.
 - o For running sum: subtract old value, add new value
 - When we want the current reading, we just divide the sum by N(average)

 Note: with RPM, the number of millisecond, (our divisor) defines the amount of averaging

Analog Outputs:

- We have GPIO(General purposes I/O) which lets us turn things on/off
- Can also use GPIO to turn components on/off
 - o E.g. activate a signal generator
- Every microcontroller has output that generate a fixed signal (5V or 3.3V)
 - If we turn on an output, we get "Full power"
 - o E.g. if we have a direct current(DC) motor, it will run at full RPM
 - Turning off the output stop the motor
 - Electronic 101(Recap): Loads
 - Microcontroller are not designed to drive heavy <u>loads</u>
 - Loads: Voltage to drive physical devices
 - Instead, microcontroller activate/deactivate power that going into the device
 - Device is powered separately
 - Microcontroller turns power on/off via transistor
 - If we turn the output on & off, we can pulse the motor
 - If we do this fast enough we can get slower speed
 - W.r.t. our motor outputs, this results in overall lower voltage (and currrent)
 - Can also do this with a magnet to vary its strength
 - Used to adjust/move valves, clutches, etc.
 - This is also how we generate sound