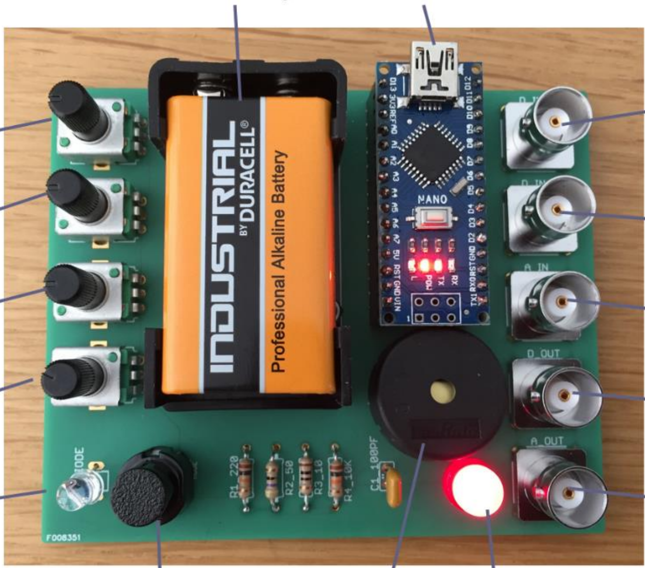
Getting started

Turn all dials to midpoint (you should feel a gentle “click”). Power the board (via USB, or via 9V Battery). The Arduino LEDs should come on. If the board is already on, press reset button on the Arduino itself (not the big button on the board). This will take a few seconds, and then put Spikeling in “reset state”. All instructions below assume you start from here.

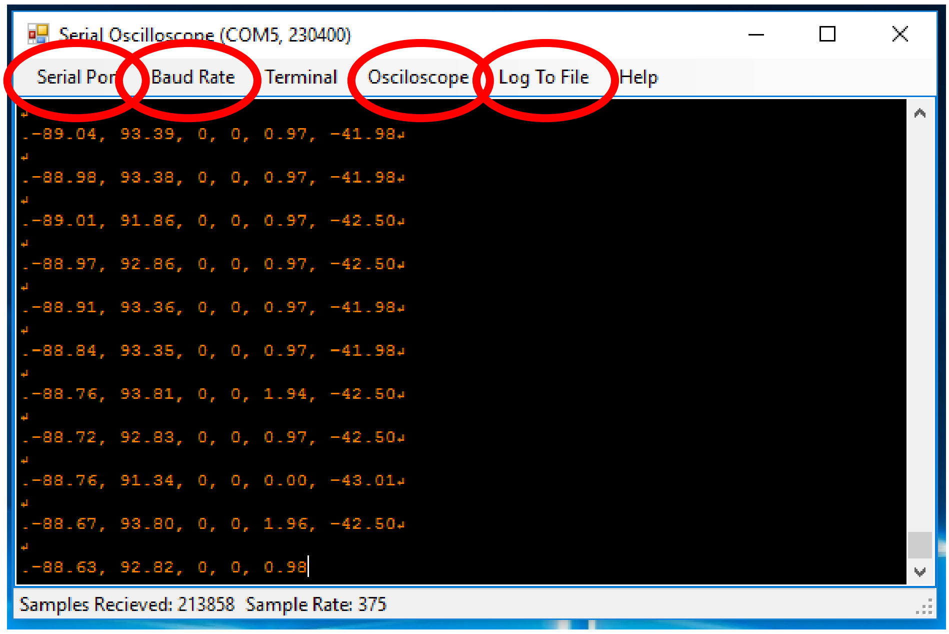
The “average textbook neuron” is a vertebrate-type spiking neuron with a resting membrane potential around -70 mV and a spike threshold around -55 mV. The reset state mimics this type of neuron.



To visualise and record the activity of Spikeling, there are several options. The best is via serial connection to a PC. Connect Spikeling to the PC via USB and start the Serial-Oscilloscope software package provided (kindly provided under creative commons from http://x-io.co.uk/serial-oscilloscope/). This lets you read all output parameters directly from the Spikeling board for display on the screen or for logging. There is also a YouTube video (https://www.youtube.com/watch?v=jgMG0UQ2_pc) demonstrating how to set up the oscilloscope.

1) Make sure the Arduino IDE is installed as you need the driver (www.Arduino.cc).
2) Declare the serial port. When you plug the USB you may see a message telling you which “COM-port” is being used. This is the “serial port” which you need to set in the menu of the serial oscilloscope. It may just give one option which is then probably the correct one. The COM port is also found under the Device manager.
3) Set the Baud Rate to 230400.
4) The window should now produce a continuous stream of numbers. These are the output values from Spikeling. To plot them, open the oscilloscope(s) under the menu. Spikeling outputs 9 parameters in parallel, but each oscilloscope window can only display 3. If you need parameters further down the list, simply open a second oscilloscope. You can separately adjust the scale of each trace on the oscilloscope by first selecting it (click the “Beam” button) and then scaling it (left-most thin arrows) and off-setting it (big red arrows).

5) To record data, click “log to file”. This will write a text file (comma and tab separated Ascii) with all numbers output from Spikeling. You can look at this output using any common data-analysis software (for example GNU-R (free), Igor Pro or Matlab).



The output parameters of Spikeling are as follows:

1) Membrane voltage V_m (in “mV“)
2) Total input current I_{total} (in arbitrary units). $I_{total} = I_{Vm} + I_{PD} + I_{Syn1} + I_{Syn2} + I_{analogIn}$
3) Stimulus state of Synapse 1 port (see below, range 0-1)
4) Synapse 1 state (see below, boolean 0-1 for nospike / spike in the input)
5) Synapse 2 state (see below, boolean 0-1 for nospike / spike in the input)
6) Total photodiode current I_{PD} (in arbitrary units)
7) Total Analog In current $I_{analogIn}$ (in arbitrary units)
8) Total Synaptic current I_{Syn} (in arbitrary units) $I_{Syn} = I_{Syn1} + I_{Syn2}$
9) System time since last reset (in microseconds)

You can also read Spikeling’s Vm output via the BNC ports using a regular oscilloscope. Digital Out only carries the spikes (TTL 5V) while Analog Out carries a version of the analog membrane voltage (in a.u.). (Note that this output is an RC-circuit filtered PWM Analog out and thus will have filtering artefacts).

Most of the time, simply displaying the 1st three outputs on oscilloscope 1 will be sufficient:

200 ms per division

Resting membrane potential

In the absence of a stimulus, the resting state Spikeling sits at -70 mV and should only spike sporadically. Resting membrane voltage (Vm) can be set indirectly with bottom-most dial, which sets a constant input current. For now, on the oscilloscope we are only interested in the membrane potential trace (the red one) and the current trace (the green one). The red LED on the board also tracks Vm, and flashes with each spike which should also be accompanied by a “click”. Electrophysiologists often connect a speaker to their recording of membrane voltage as a helpful way to know when a spiking neuron is near the electrode, and to get a direct audio feedback of what the neuron might respond to.

Task 1: What happens when you increase or decrease the static input current?

*You should observe that increasing the static input current drives Vm towards and beyond spike threshold. As you keep driving Vm upwards, you will elicit progressively higher spike rates. This is the simplest of all **neuronal codes** - the intensity of a stimulus (here, simply the increased input current) is **encoded** in the frequency of spikes. Imagine you are the postsynaptic neuron and all you see is this spike pattern – you could easily infer from seeing more spikes in close succession that probably the input to the presynaptic neuron has increased. Most spiking neurons use this **rate code** to signal input intensity.*

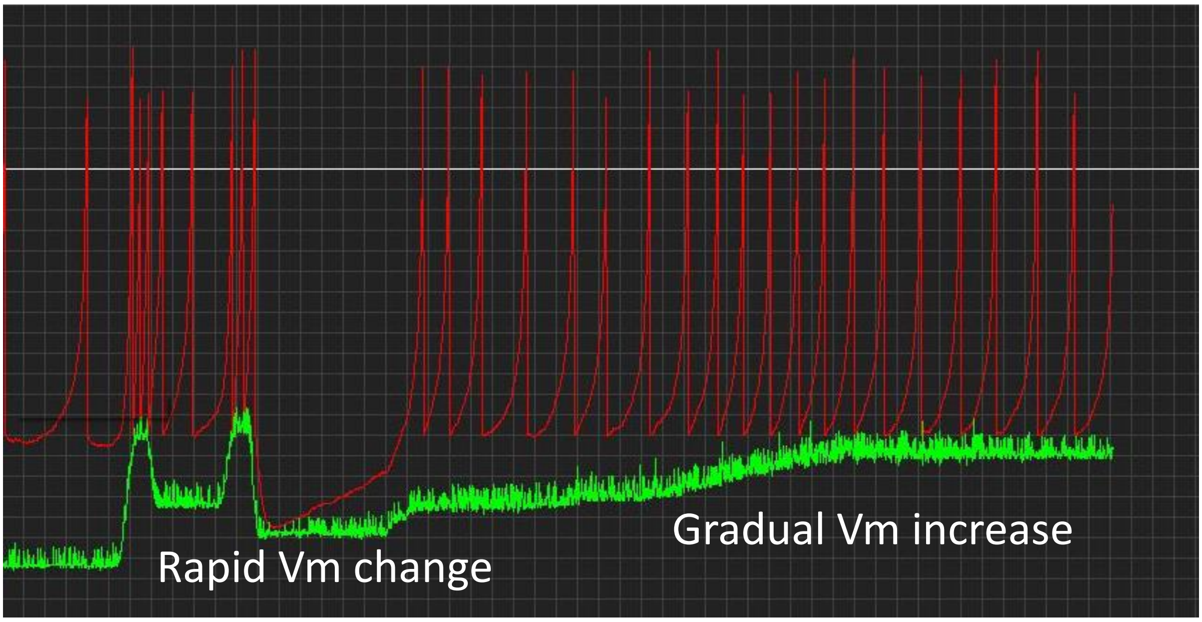
*On the screen, note that each spike is preceded by a shallow rise in Vm and followed by a brief dip below starting levels. This dip is the refractory period of the neuron. During this time, generating another spike is particularly difficult. At the extreme lowpoint, it is impossible as in a real spiking neuron, the sodium channels are blocked (not just closed) – as such, you cannot generate an infinitely fast series of spikes – this **absolute refractory time**, (and the duration of the spike itself which is typically 1-2 ms) defines this limit. In an average neuron, the absolute refractory period is a few milliseconds, and thus puts the maximal spike rate of most neurons at 100-200 Hz. Some specialised neurons can go a bit higher, but kHz range is out of question. This means that using a single spiking neuron, it is impossible to faithfully encode a time-varying stimulus above this frequency. However, there are a few tricks around this problem that the nervous system can use. We will pick up on this point later.*

Task 2: What happens when you dial current up and then wait a few seconds?

*If you drive up input current and leave it there for a few seconds, you should observe that spike rate first increases, but then will taper off to some new basal rate of activity which will be higher than the original rate (rate code), but lower than the peak rate. This is an example of **adaptation**. Neurons respond to a change in the input not only by firing more or fewer spikes, but in addition by adjusting their sensitivity to further changes based on recent stimulus history. This is a fundamental property of neurons that allows them to extend their operating range.*

Task 3: Does a rapid and a slow current increase give the same result?

*As you increase input current slowly or rapidly, you should observe that you can reach different peak spike rates. A rapid increase in input current is a much more effective way to trigger multiple spikes in close succession. This is again because of adaptation. If you change input current fast enough, the neuron does not have time to adapt and therefore fires vigorously at first. If you change input current slowly enough, you should be able to drive it quite high without eliciting any extra spikes. This means that not only the absolute level of a stimulus can be encoded by a neuron, by also the rate of change. Note that this creates ambiguity in the code, which is one important reason for the need of **parallelisatisation**. This means that if you want* *to read both absolute levels of a stimulus and its rate of change, you may need two neurons with different properties.*



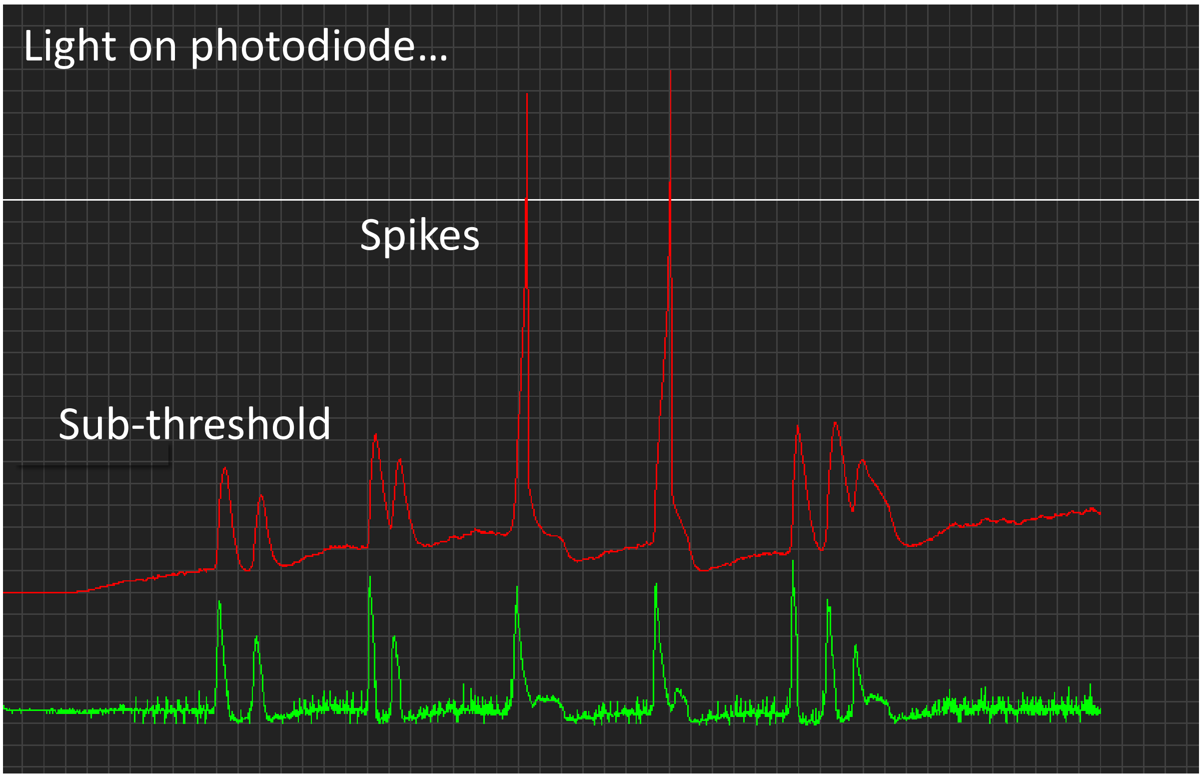
NB: The fact that the speed of change in the input is encoded in a neuron's firing also means that that spike thresholds are not fixed. Depending how quickly you stimulate a neuron, it can start firing at different Vm values!

External stimulation: Light

Spikeling has a built-in photodiode – the clear dome-shaped object in the lower left corner. This functions like a "mini solar-panel". If you shine light at it, it generates a tiny voltage, and Spikeling is programmed to react to this voltage. Increasing the amount of light hitting the photodiode drives a depolarising current, just like the static input current dial did (above).

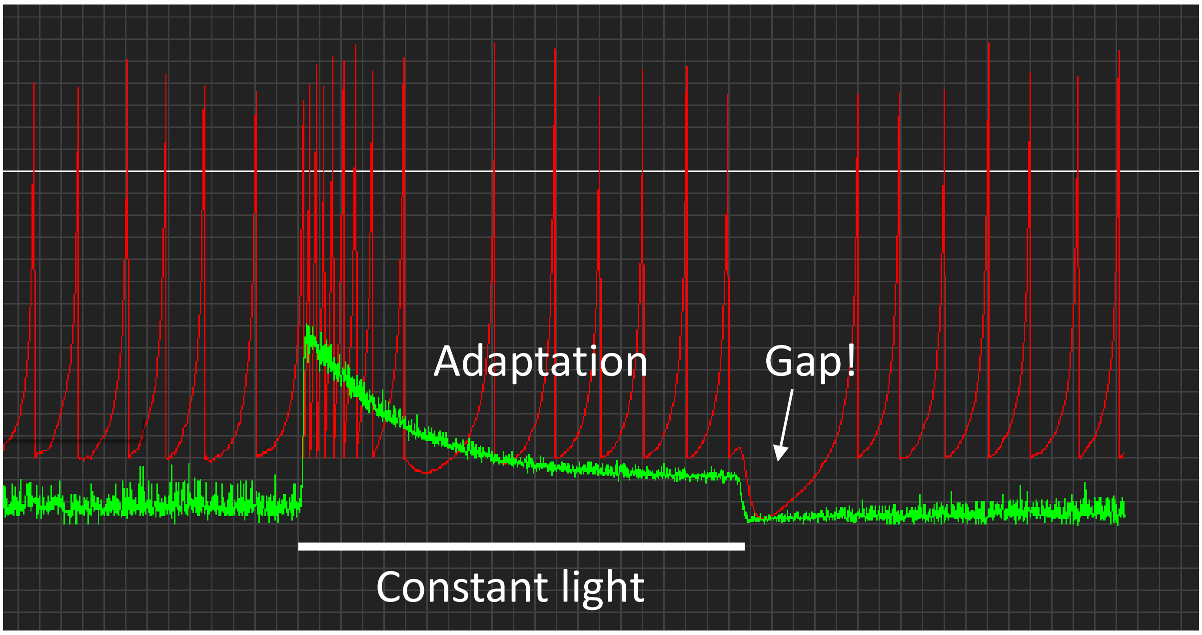
Task 4: Shine some light at the photodiode e.g. using a torch or by holding it into the room light. Observe both the current (green) and the voltage trace (red). Can you get the cell to spike in response to light? Does it spike every time you hold it into the light?

Shining light at the photodiode should drive a brief increase of the driving current, which will be mirrored by the membrane voltage. If the light intensity is high enough, you should be able to generate spike(s). However, if the light is weak, or presented only for a very short amount of time, you will probably not drive a spike. This is an example of how neurons can use a spike threshold to only report the presence of a stimulus if it is of a certain amplitude and minimum duration. However, do note that even if there is no spike generated, the membrane voltage will still react to most changes in the light. This ***"sub-threshold" activity*** *is fundamental to many neuronal computations. We will return to this point later.*



Task 5: Turn the static input current dial up so that Spikeling generates a few spikes per second, and then shine some constant light at it. You should observe that the spike rate increases a lot at first, but then settles to an intermediate spike rate (just as before when you increased just the Vm dial). Now, suddenly remove the light. What happens?

You should observe that when you remove the light, membrane voltage will drop not just back to baseline levels, but below baseline for a short period of time. As it drops below baseline, it will probably result in a brief gap in spikes. This is an example of how a sudden absence in a stimulus that the system has adapted to can be encoded by the neuron by the absence of *expected spikes. However,* ***the salience of this code is low****. For a postsynaptic neuron reading this signal, it is much easier to respond to the presence of an unexpected spike, rather than the absence of an expected one. (How could you turn the absence of an expected signal into the presence of an unexpected one?)*



Switching spike-modes

Spikeling comes with multiple pre-programmed behaviours, which can be cycled through with the on-board button (the big black one). If you press the button from reset state, a little LED on the Arduino should light up twice. This means that Spikeling is now in “Mode 2”. If you press it again, it should blink 3 times (Mode 3) and so on. If you get to the end (Mode 8), it will cycle back to 1. You can always jump to 1 by resetting the Arduino.

The pre-programmed modes are as follows:

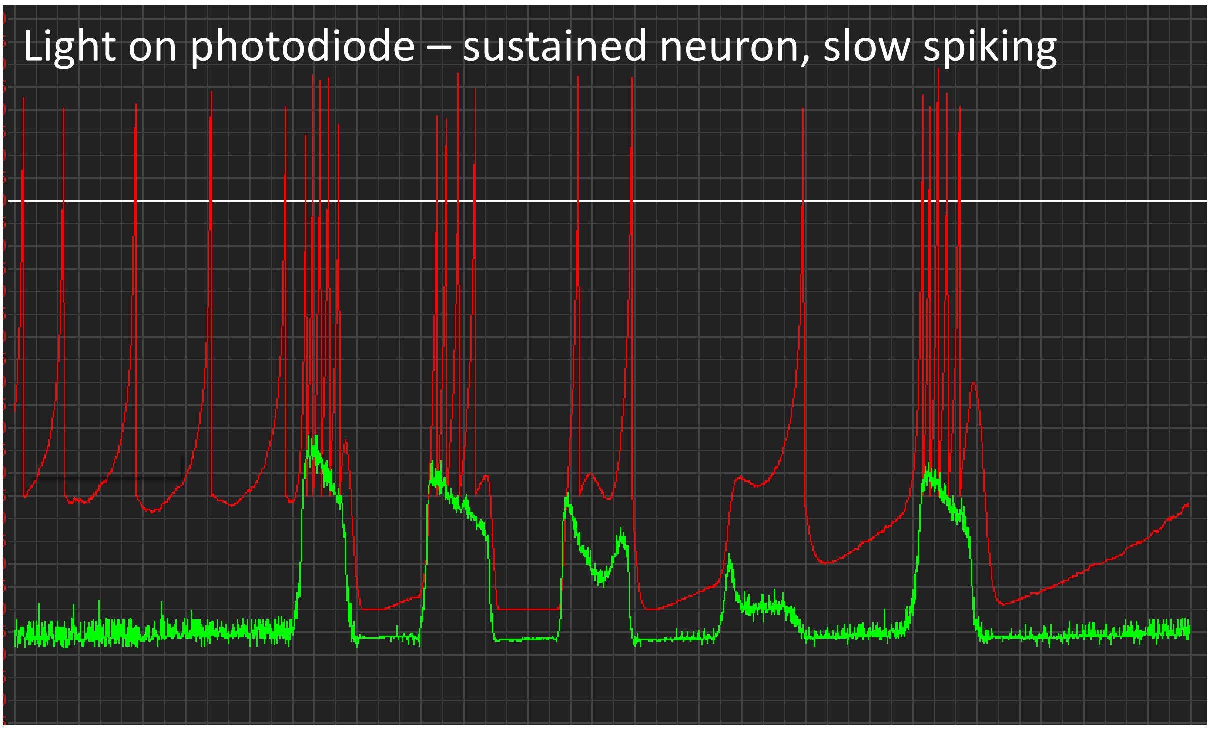
1) Regular spiking neuron, slow adapting photodiode
2) Regular spiking neuron, fast adapting photodiode
3) Fast spiking neuron, slow adapting photodiode
4) Fast spiking neuron, fast adapting photodiode
5) As 3, but synapse 1 port switched to output delivering 50% duty cycle steps
6) As 4, but synapse 1 port switched to output delivering 50% duty cycle steps
7) As 3, but synapse 1 port switched to output delivering 50 Hz binary noise
8) As 4, but synapse 1 port switched to output delivering 50 Hz binary noise

For now, let’s just focus on the first four modes. Compare the above exercises on resting membrane voltage and photodiode stimulation in different spike modes.

Task 6: Comparing Modes 1 and 2, what is the difference when you pass a torch over the diode?

The spiking behaviour of the neuron is identical, but the speed at which the response to photodiode stimulation decays is very different. If you continuously shine light at Spikeling in Mode 1, it will adapt slowly. If you do the same for Mode 2, it will adapt a lot faster and to a stronger degree. This is an example of a ***“transient”*** *(Mode 2) and a* ***“sustained”*** *neuron (Mode 1). In the nervous system the transience of a neuron is one fundamental ingredient in generating different functions. For example, transient neurons are usually good at* ***encoding the onset of a stimulus****, but not very useful in signalling when that stimulus stops. In contrast, the sustained neuron encodes both events quite reliably, but the* ***energetic cost is higher*** *(it needs more spikes) and the* ***information content per spike is lower****. Neurons are amongst the energetically most costly cells in the body. They are also amongst the most fragile if energy supply is low (for example during a stroke!). Most neurons will die after even a few seconds to minutes of oxygen deprivation. Accordingly, when considering neurons and neuronal networks, it can often be instructive to consider the energy costs associated with a particular computation. If you can implement the same computation using fewer spikes, or fewer neurons, that is probably a good thing, and most of the time it is what the brain will have evolved to do. Accordingly, it is not always about setting up the “best” computation, often it is about building the cheapest system that still works with adequate reliability.*

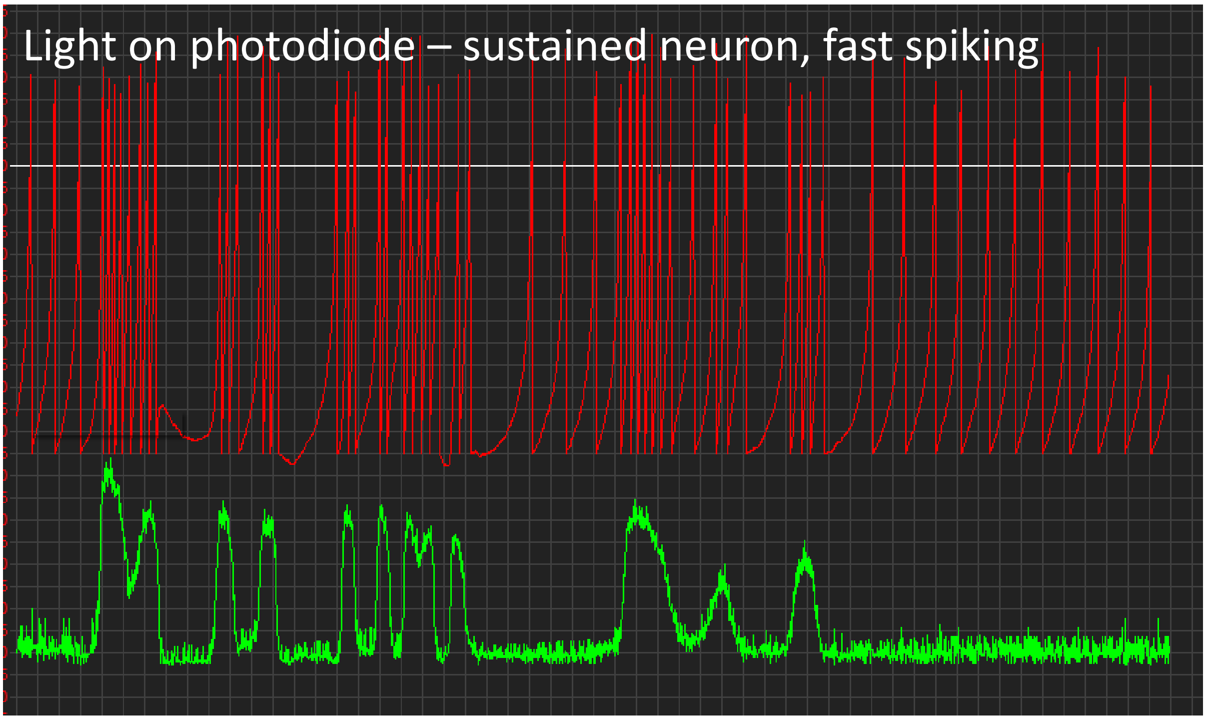
Another reason for using a transient neuron is that after it signals the start of an event, it is rapidly “ready” to signal the start of another event. If a transient neuron spikes twice, it probably means there were two events. If a sustained neuron spikes twice, it could mean there are 2 events or that there was one event which is still ongoing. Accordingly, the sustained neuron’s ***code*** *is usually more* ***ambiguous****. Ambiguity in coding is almost always a bad thing. It reduces the information content per spike (costly!) and it usually means you need additional neurons to resolve the ambiguity (also costly).*



Task 7: Compare Modes 1 and 3 by playing with resting membrane voltage and the photodiode. What is the difference?

Now, in both cases the photodiode is sustained, but the neuron itself behaves a bit differently. In Mode 3, the spike threshold is lower and the refractory period following a spike is shorter. This neuron will do most things that the Mode 1 neuron will do, but it will use more spikes. While energetically costly, sometimes you need more spikes. For example, for a rate code to encode as much information as possible, the bigger the rate-range is that the neuron can cover, the more ***finely resolved can you encode signals*** *using the frequency of spikes. One often used example here are projection neurons of auditory systems.*

Another advantage of using a neuron that can reach high spike rates is that ***negative coding*** *becomes more useful. If the basal firing rate of a neuron is high, a reduction in this rate (for example due to the sudden absence of a stimulus, or due to the addition of an inhibitory input) can still be readily read out by postsynaptic neurons. This is easily illustrated if you repeat an experiment from above: Set the basal spike rate to a few spikes per seconds using the static input current dial, then shine a constant light at the photodiode and then remove the light. Each time you remove the light, the high spike rate should drop to a new, lower spike rate, and this change should be much more obvious is you use the fast spiking neuron from Mode 3 compared to the regular spiking Mode 1 neuron.*


Light on photodiode – sustained neuron, fast spiking

The “self-stimulator”

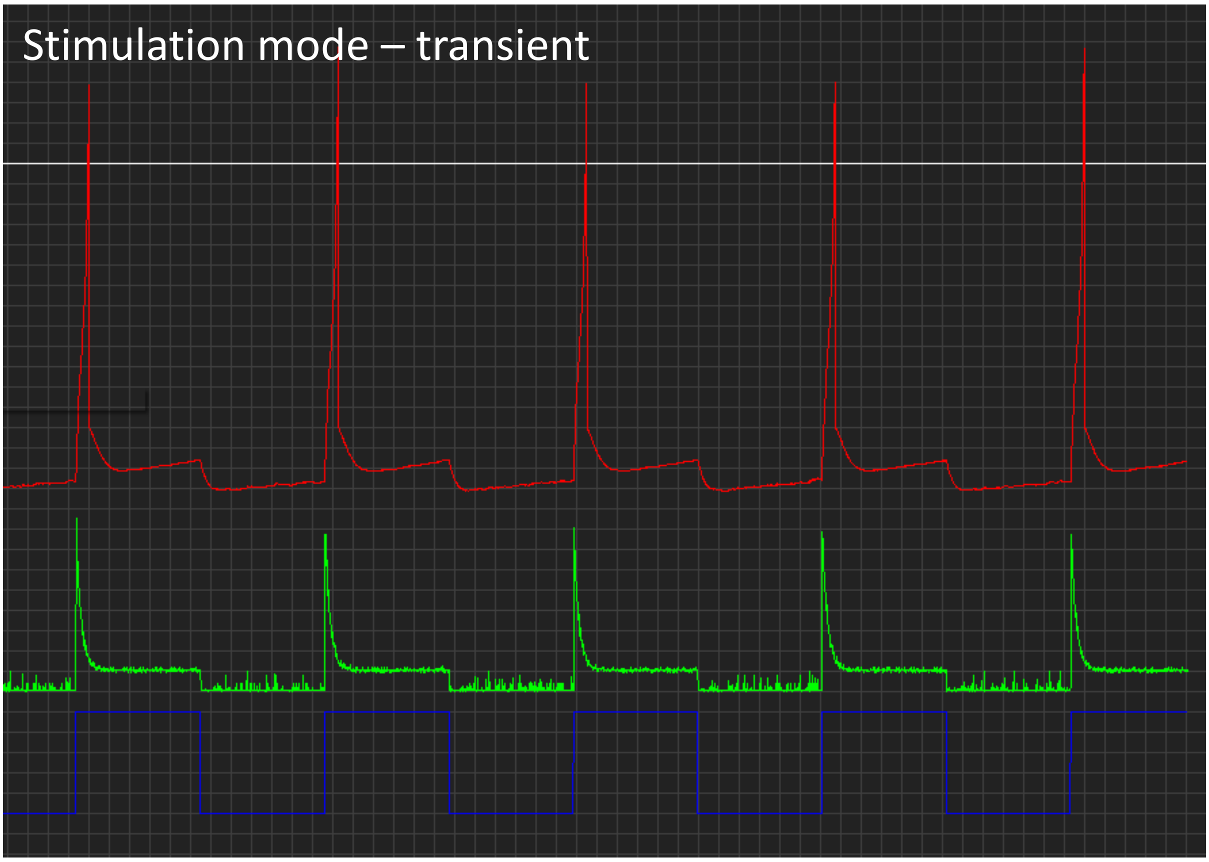
From the exercises with the torch, you will have noticed that it is difficult to exactly reproduce the exact same stimulus time and time again. To help you with this, Spikeling comes with an option to “stimulate itself”. If you put it into Modes 5-8, the BNC connector labelled “Synapse 1” is internally configured as an output port which delivers 5V pulses at different intervals depending on the mode. Connect a BNC cable to this port, and on the other end connect the extra BNC-cap with an LED attached. You should now see that the LED lights up in regular intervals (Modes 5,6) or flicker randomly at 50 Hz (Modes 7,8). Attach this LED on top of the photodiode (e.g. with a bit of tape or a ring of paper) such that whenever the stimulus LED lights up you can clearly see an increase in the input current on the oscilloscope. In addition, configure the 3rd trace on the oscilloscope (blue trace) so that you can see it switch between 0 and 1 and to indicate the state of the stimulus.

You should now have 3 traces on the screen: The red membrane potential, the green input current, and the blue stimulus. Every time the stimulus is high, the green trace and therefore the red trace should both respond with an increase in turn. This set-up will let you explore the response properties of Spikeling in a more controlled manner. (Note that all Stimulus modes (5-8) are configured to use the “fast spiking mode” for either the transient or sustained photodiode setting).

For now, let’s focus on Modes 5 and 6. For both, the stimulus LED switches on and off at regular intervals, such that it is on and off exactly half of the time (the technical term is “**50% duty cycle**”). If you now turn the top-most dial away from resting mid-position, the frequency of this flicker will change accordingly.

Task 8. Comparing Modes 5 and 6, observe the responses to the ~1 s steps of light that are elicited with the speed dial set to mid-point. Now, play with the static input current in each mode. Can you get Spikeling to reliably spike exactly once during stimulus onset in transient mode (6)? Can you get Spikeling to reliably spike at the offset in sustained mode (5)?

You probably found that it is quite easy to set the transient neuron to encode the onset with just one spike, but that it is very hard to find a suitable set-point for the sustained mode to signal the stimulus offset in a time-accurate manner. This comes back to a point raised before. It is much easier to signal the start of an excitatory event, compared to the end of it.

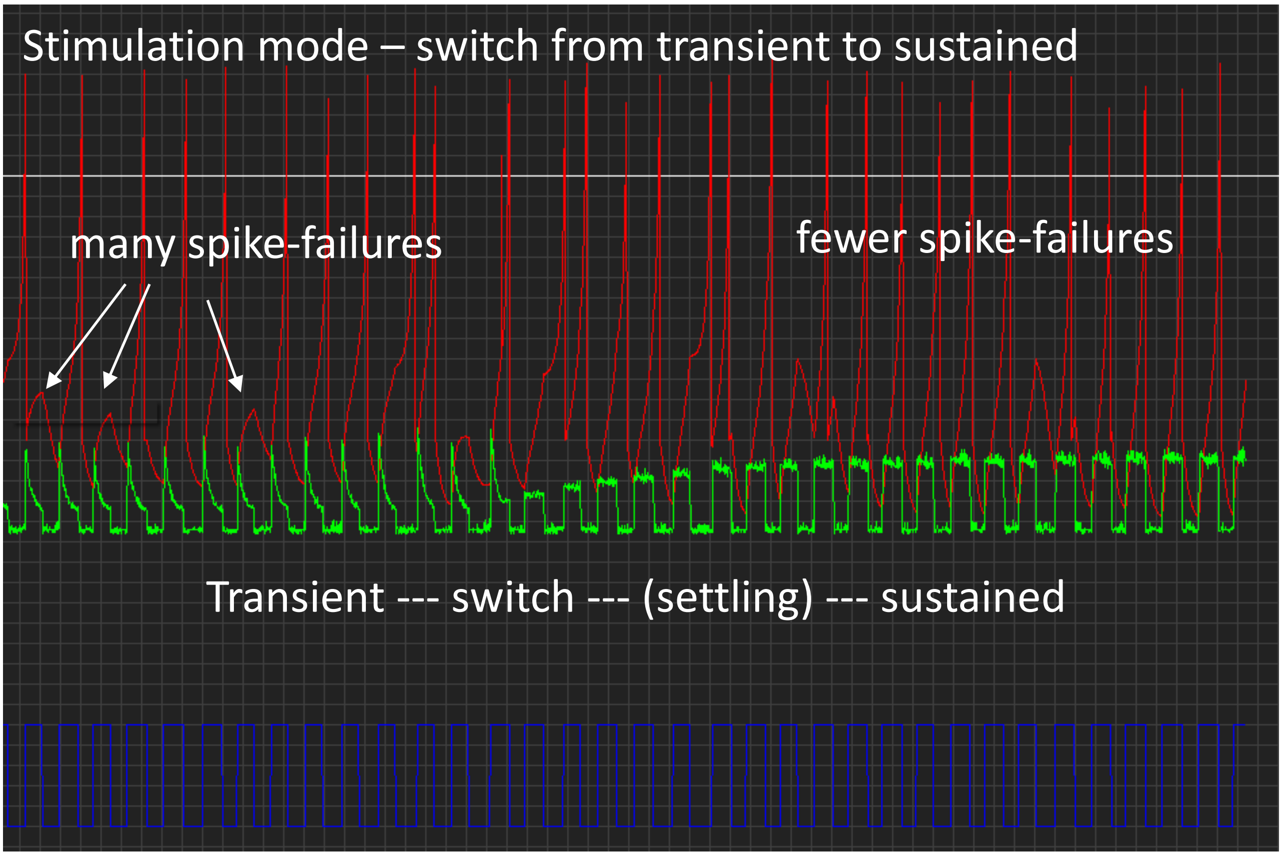

Stimulation mode – transient


Stimulation mode – sustained

-

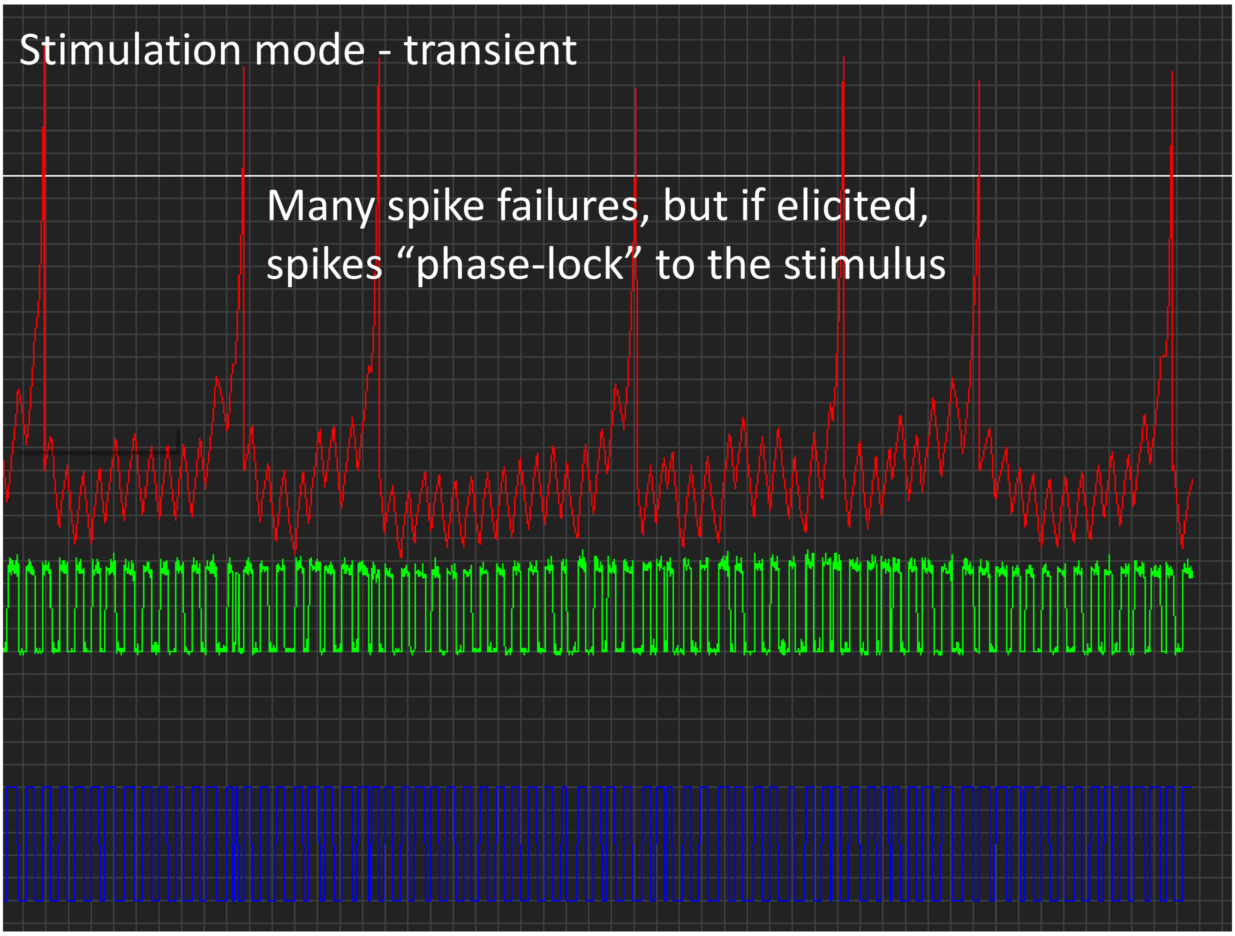
Task 9: Now, start playing with the speed of the stimulus while adjusting the static input current as required. How fast can you go with either the transient or the sustained neuron until you start observing spike failures?

*You will probably find that both neurons do a reasonable job until about ~2-3 Hz stimulation in encoding each even with a spike, but afterwards start of fail spiking every now and then despite the presence of a stimulus. As Spikeling is set-up, it will probably do a little better using the sustained neuron. This is no inevitable – if one were to change the spiking parameters a bit, it would be possible to make the transient neuron more reliable. More importantly, the photodiode, the spike mode settings and the static input current all come into play when determining the limit at which a neuron can follow a time varying stimulus. As such, **"tuning" a neuron**'s response preference to a desired input requires setting a myriad of properties. The nervous system utilises this to tune each neuron to a specific range of input statistics, and, indeed, to adjust these settings as the task at hand requires.*



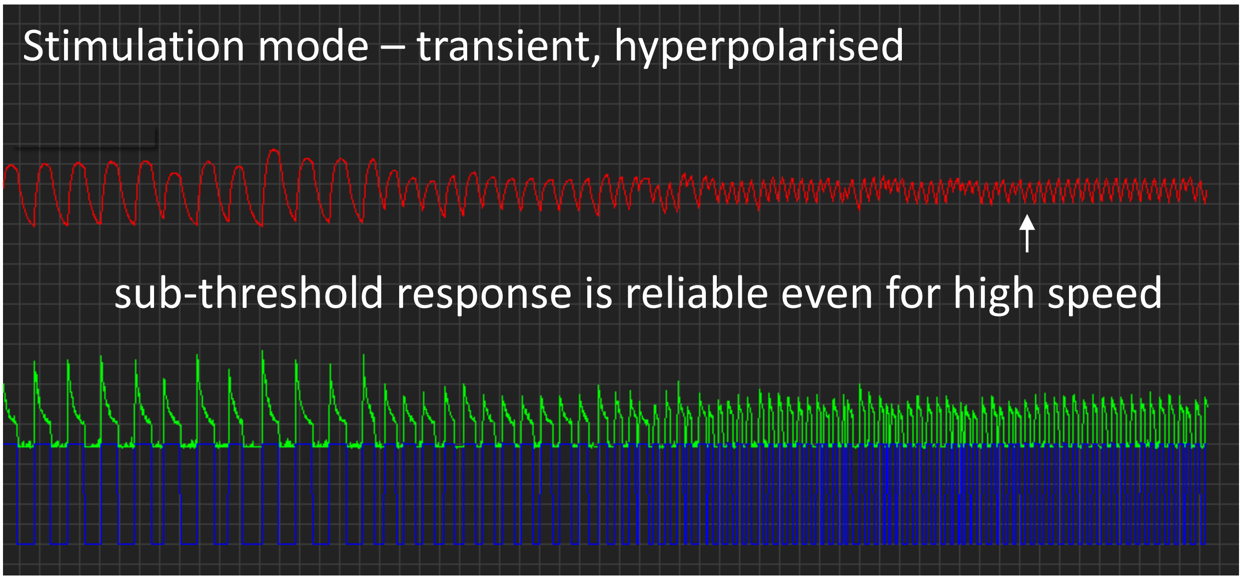
Task 10. Keep increasing the speed of the stimulus until it nearly fails to generate any pulses at all, and set the static input current such that it only spikes occasionally. What do these spikes encode?

*If you set Spikeling into a regime where most stimuli do not trigger a spike, you will notice that nonetheless each stimulus drives a clear subthreshold response. If you now look carefully at the timing of each spike that is elicited, you will likely find that this time is fairly well "**phase-locked**" to the onset of a stimulus phase. In isolation, this is not a particularly useful property of neurons. But imagine you had 10s such neurons, each spiking unreliably every 5-10 stimulus phases. If you would sum the input from these 10 neurons using a larger,* *postsynaptic neuron, you could fully reconstruct the original stimulus train even though no single one input neuron encodes that information. This is called a "**volley code**", and is one of the most important tricks that neuronal networks use to encode a fast time-varying stimulus that exceeds the speed that any one neuron can encode. For example, in auditory systems, the stimulus frequency can easily reach the kHz range which no neuron can possibly follow, as discussed above. Nonetheless, if you record any one neuron's firing in response to such a stimulus you will probably find that the timing of each spike is exquisitely well phase locked to the stimulus, such that if you had a few 100 of these neurons you could precisely reconstruct the original input.*



Task 11. Decrease the static input current until no more spikes are triggered. Observe the membrane potential as you increase the stimulus speed. Can you get Vm to fail following the stimulus?

*You will probably find that up to the speed that Spikeling can stimulate (~10 Hz), Vm comfortably tracks the stimulus (while spikes do not). This is a fundamental property of neurons. The subthreshold response of a neuron is almost always better at following stimuli than the spike response. As discussed above, eliciting a spike and resetting the neuron to be ready to fire the next spike takes time (in Spikeling, this takes >10 ms, some real neurons can reset after 1-2 ms but not faster). If you do not need to generate a spike, this limitation is dramatically reduced, and you can follow stimuli much better, and much more accurately (also in amplitude). And indeed, **not all neurons spike**! In fact, spiking is an incredibly costly and inefficient way of conveying information, so if neurons can avoid using spikes, they will. For example, one fundamental reason to use spikes is to rapidly cover large distances. If you have* *a big brain, you need to use a spike to send a signal from one end to the other. Depending on a neuron's electrotonic properties, a **graded signal** would just decay with distance and be unrecognisable ~100 microns from the origin. As such, the regenerative nature of spike propagation makes it a good choice for this task. However, as you generate the spike, you truncate the message, both in time (refractory period) and in amplitude (threshold). So, if the distance to be covered is small, using spikes is usually avoided. For example, C.elegans barely uses any spike at all. The animal is so small that all distances are small. Similarly, some neurons in the Drosophila brain don't spike at all. Other use a **"mix" of spiking and graded processing**. In vertebrates, the same thing happens in some neurons. For example, in the retina, only about half of the neurons use spikes. Photoreceptors, for example, generally don't use spikes. Ganglion cells, on the other hand, connect the eye to the brain via the optic nerve – so there is no way round it, those neurons have to spike.*



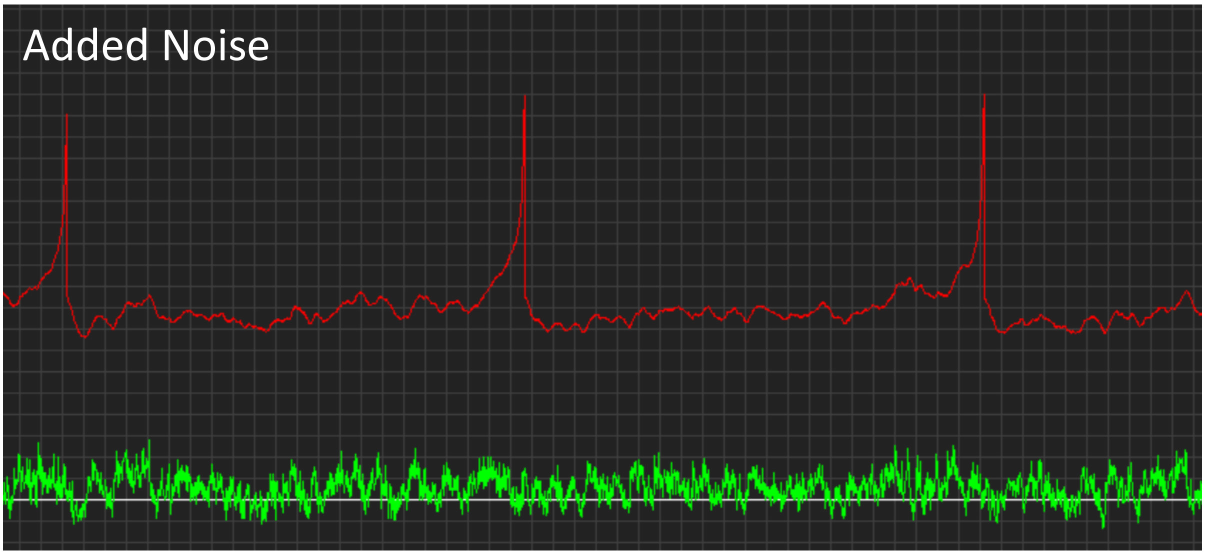
Finally, modes 7 and 8 provide fast flicker. This is intended for estimation of "linear filters" of Spikeling. This is for advanced users only, and we will come back to this point later.

Noise

So far, we have been working at low noise. However, real neurons are noisy. Sources of noise include synaptic noise, receptor noise, thermal noise or even noise associated with the physical world that an animal inhabits. Accordingly, one major challenge that all neurons face to at least some degree is how to detect the meaningful "**signal**" in the background of meaningless "**noise**". Spikeling provides the option to add different level noise to any operation – for this, turn the second dial from the bottom ("Analog In / Noise"). Note that this dial is a doubly-used control – it sets the noise level but also acts as a scaling factor for the Analog input port. If only one of these is desired, either function can be switched off in the Arduino code in the top (advanced users only). For now, since there is no Analog input connected, turning the dial will only affect noise levels. At mid-point (reset-state) and below that, no noise is added, but above that position increasing the dial will linearly increase a "noisy current" which adds to the total current shown in the green trace.

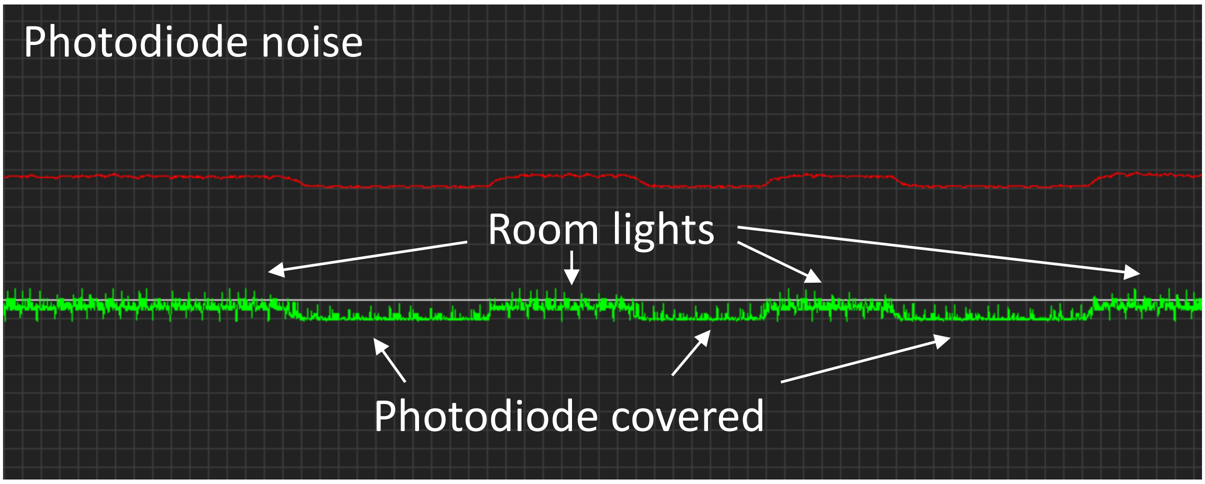
Task 12. Turn up the noise dial, while keeping the mean input current roughly centred around 0 (use static input current dial to offset this if necessary). What happens to membrane voltage as you add more and more noise?

*At first, increasing the noisy current will only affect the baseline noise of membrane voltage, which will resemble a **low-pass filtered version** of the input noise. This is the first "trick" used by all neurons to dampen high frequency components in the noise – the membrane potential simply cannot track the fastest transients in the input current and therefore "automatically" filters them out. As you keep increasing the noise level, you should be able to elicit spikes. These noise-driven spikes are almost always a bad thing. The do not convey any information (thus wasted energy), and to make things worse, they confound any message that the neuron is aiming to encode using spikes. Accordingly, neurons often aim to keep the spike threshold high enough such that the noise they have to deal with by itself very rarely triggers a spike. Accordingly, **spike generation can be used as a powerful noise filter**.*



Task 13. Turn the noise dial back down, and look at the baseline. Now cover the photodiode (e.g. with your hand). What happens?

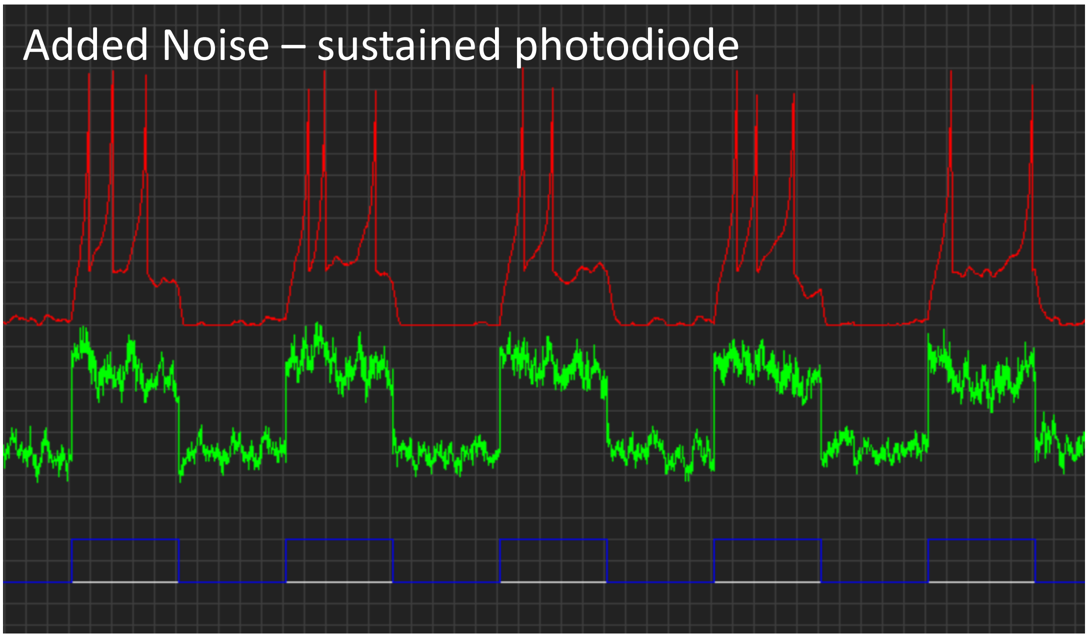
You will probably find that when you cover the photodiode, the baseline noise that we had all along in the above exercises decreases (but does not disappear). Clearly, the photodiode is introducing noise to the system! In this case this is high frequency noise that largely gets smoothed at the level of membrane potential. All sensory processes, whether electronic as here, or in biology, necessarily introduce noise - simply because the apparatus to pick up the desired physical stimulus is never perfect. In vision, for example, the stimulus (photons) is absorbed inside an opsin-type protein by a chromophore. As the photon arrives, it photoisomerises the chromophore and thereby sets a biochemical cascade into action that ultimately results in the opening of ion channels. However, the isomerisation event can also occur "spontaneously" – or rather driven by heat. As such, all sensory systems are noisy, and the nervous system has evolved a wide range of little tricks to overcome this problem. Here, spike generation or the membrane voltage filtering high frequency noise are but two examples. Others include summation (having multiple neurons signal the same thing and then adding

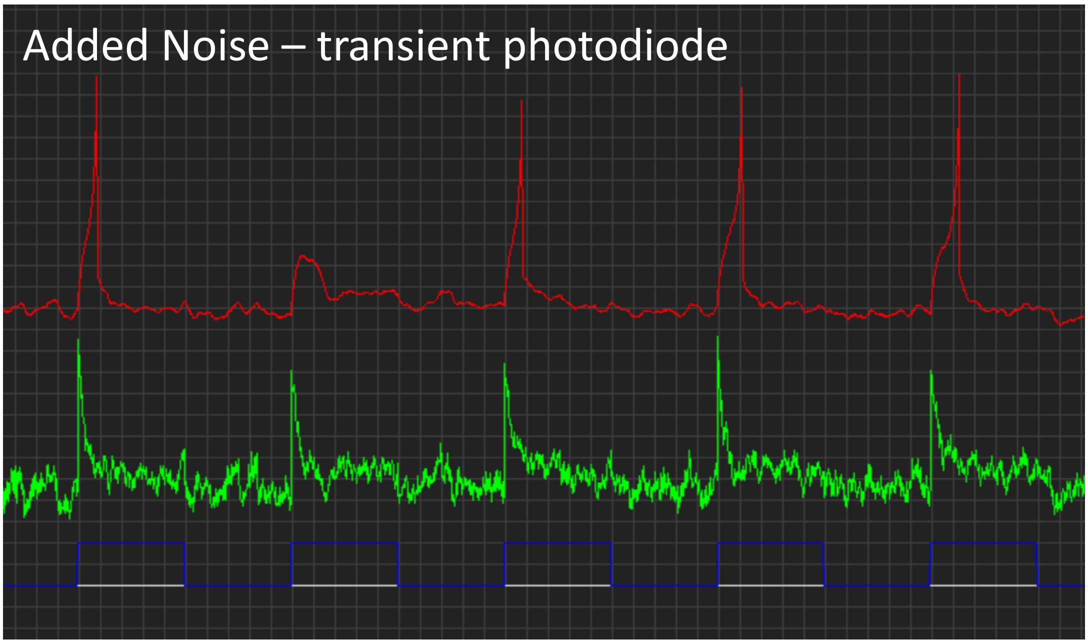


the signal up in a postsynaptic neuron) and temporal "smearing" (e.g. by using slow receptor cascades) are two further examples. There are many more.

Task 14. Compare stimulus Modes 5 and 6, and how each deals with noise. Which one would you use to reliably trigger spikes in response to each stimulus? Which one would you choose to make sure spikes are only elicited at the start of a stimulus (even if sometimes they fail)?

You will probably find that the sustained neuron (Mode 5) will be quite good at reliably firing at least one spike in response to each stimulus. However, the number and timing of each spike per stimulus cycle will likely vary a lot. In contrast, the transient neuron might fail every now and then to report the presence of the stimulus, but if there is a spike, it almost certainly means that it was preceded by a stimulus. As such, spikes from the transient neuron are highly informative, but unreliable. How can nervous systems make this mechanism more reliable?


Added Noise – sustained photodiode


Added Noise – transient photodiode

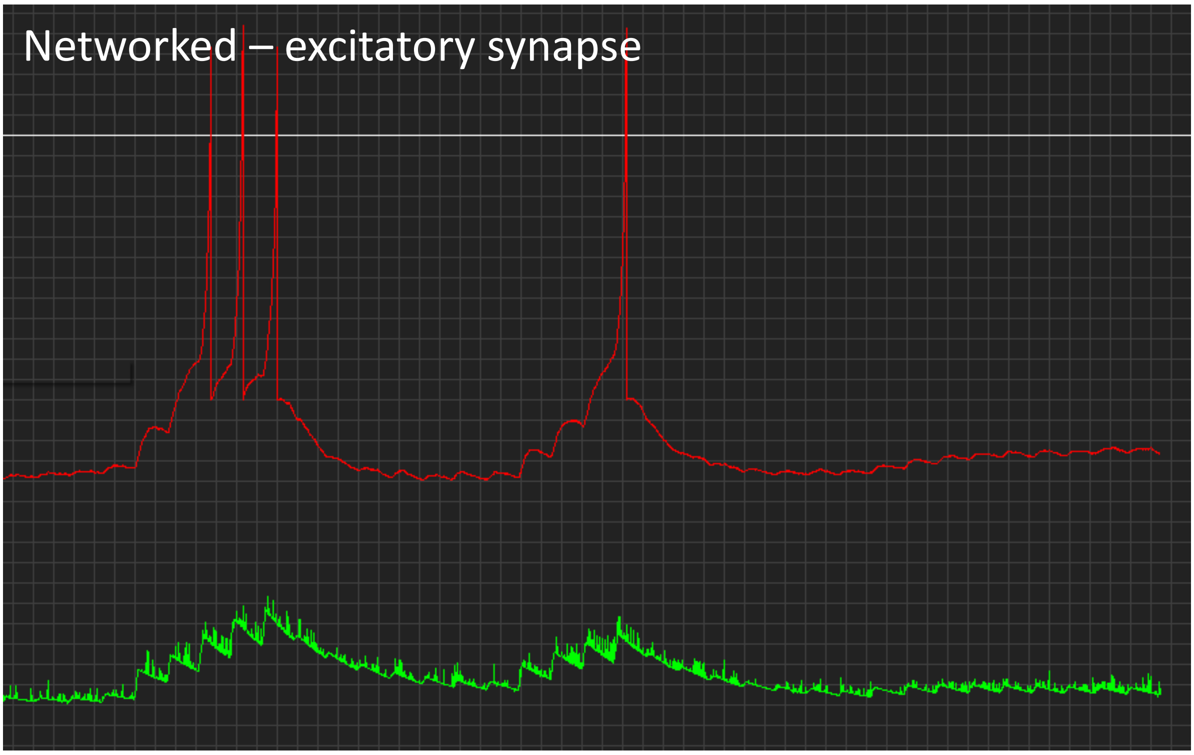
Building networks

Multiple Spikelings can talk to each other using the Digital out and Synapse 1 and Synapse 2 BNC ports. The idea is that the Digital out port of one Spikeling (2nd from bottom) conveys only spike events, which can be fed into the Synapse 1 or 2 input ports of a second Spikeling (top 2 ports). The gain of each synapse can be regulated using the corresponding dial on the other side of the board. At reset state, the gain is zero (so the synapse will not do anything if it receives a spike). If you turn the dial up, you get an excitatory synapse. If you turn it below midpoint, you get an inhibitory one.

Try this now: Reset two Spikelings and take a BNC lead to connect the Digi out port of one (presynaptic) to the Synapse 2 in port of the second. Connect the second Spikeling (postsynaptic) to the PC via the USB cable so that we can read its activity on the oscilloscope. The presynaptic Spikeling does not need to be connected– but it does need to be powered, either using a 9V battery or by plugging the USB into a power socket).

Task 15. In this configuration (above) increase the static input current on the presynaptic neuron so that it continuously fires a few spikes per second. Now observe what happens to the postsynaptic neuron as you turn up the synapse 2 dial to increase its gain.

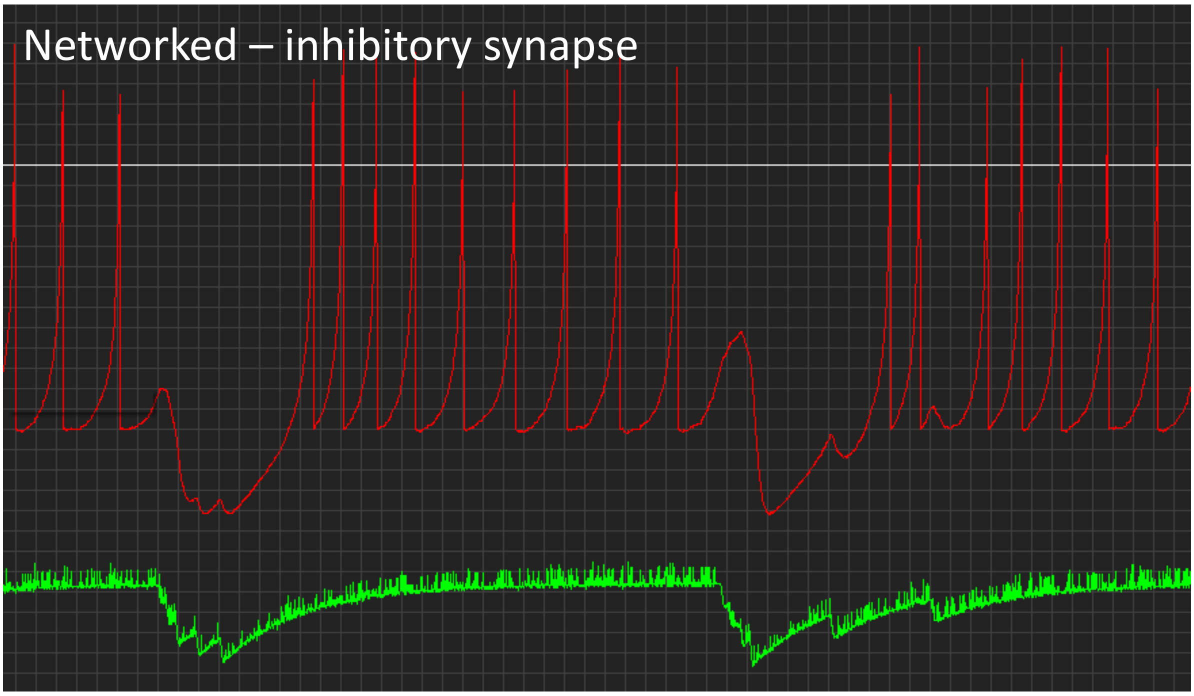
You should observe that if the dial is at midpoint, nothing much happens – the spikes from the presynaptic Spikeling are still received, but they do not drive any current in the postsynaptic neuron. (Note that you can see the incoming spikes if you open a second oscilloscope – Synapse 2 spikes are on Channel 5, so the green channel of the second osci). Now as you increase the gain you should see that each spike triggers small depolarising current which decays back to baseline within a few 100 ms. If spikes come in a sufficient rate, ***this input current will start to integrate between successive incoming spikes*** *to further depolarise the cell. If it reaches threshold, spikes should be triggered.* ***The fact that the synaptic current outlasts the duration of the incoming spike is a fundamental ingredient to neuronal processing****. It means neurons can "count and calculate" using synapses. For example,*



suppose for a particular computation is was important to know that there was not just one, but two spikes fired within close succession. This could be easily computed by setting the excitatory gain of the connecting synapse(s) such that a single spike does not drive the postsynaptic neuron to threshold, but that if two spikes arrive in close succession threshold is reached and thus the neuron fires. The same logic also applies across different synapses, which allows the implementation of ***coincidence detection****. We will come back to this later.*

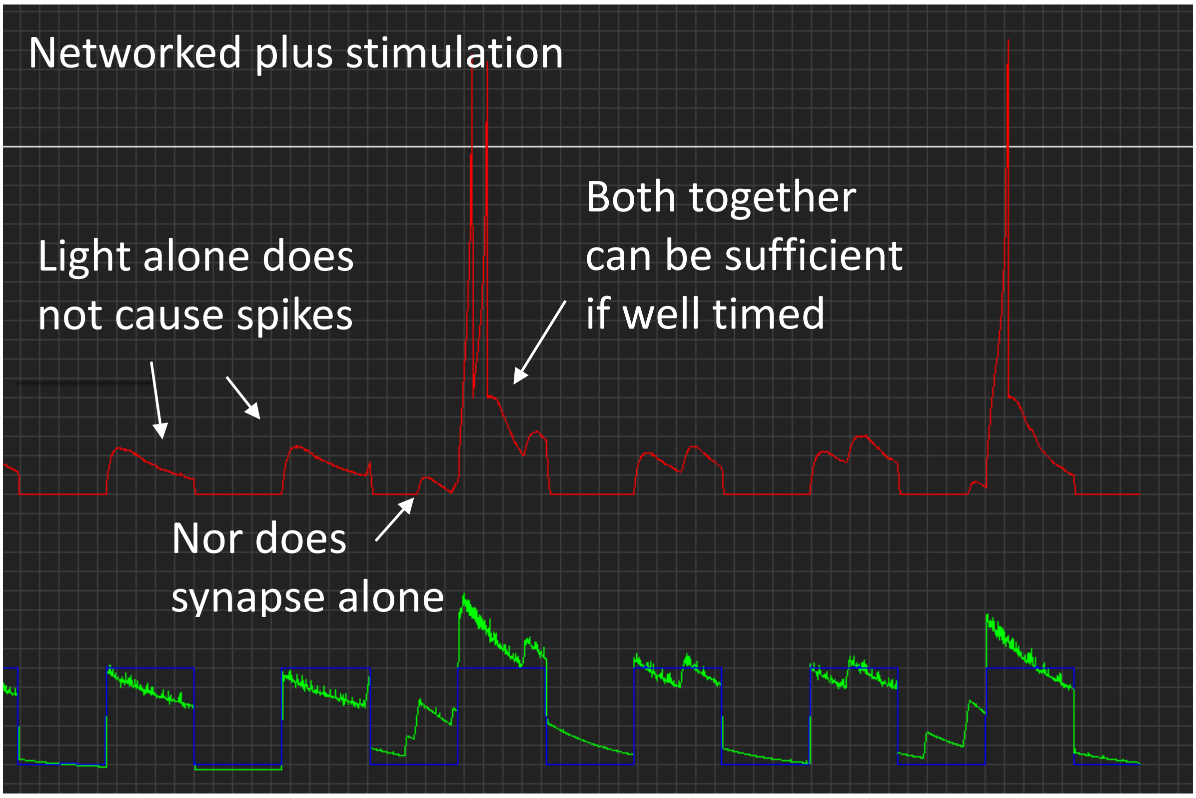
Task 16. Now turn the Synapse 2 dial below midpoint. What happens?

You should observe that now each presynaptic spike drives a hyperpolarising current in the postsynaptic neuron. This is intended to mimic an inhibitory connection. In case of Spikeling, this connection is programmed to be mirror symmetric to the excitatory connection in every way. However, in reality the gain and time courses of synaptic events can vary dramatically and are another fundamental ingredient to building computational networks. For the purpose of this tutorial, the time-course of the synapses is fixed. However, advanced users can change this in the annotated Arduino code.



Task 17. Put the postsynaptic neuron into Mode 5 (Stimulator mode, sustained), position the stimulus LED above the photodiode as before, but now hyperpolarise it using the static input dial such that the LED alone does not elicit spikes. On the presynaptic neuron, make sure that it does not fire spikes at rest, and configure its input to be weakly excitatory. Now take a torch to stimulate the photodiode on the presynaptic neuron and thereby make it spike. Can you get the postsynaptic neuron to spike?

Neither the stimulus light nor the synaptic excitation alone should be sufficient to drive a spike in the postsynaptic neuron. However, if you make the presynaptic neuron spike using the torch at the same time as the stimulus comes on, the two excitatory inputs will summate and you should be able to reach threshold. This is an example of a ***coincidence detector****. If the* *postsynaptic neuron spikes, it means that both the stimulus (stimulator) AND the presynaptic stimulus (torch) were both active at the same time.*



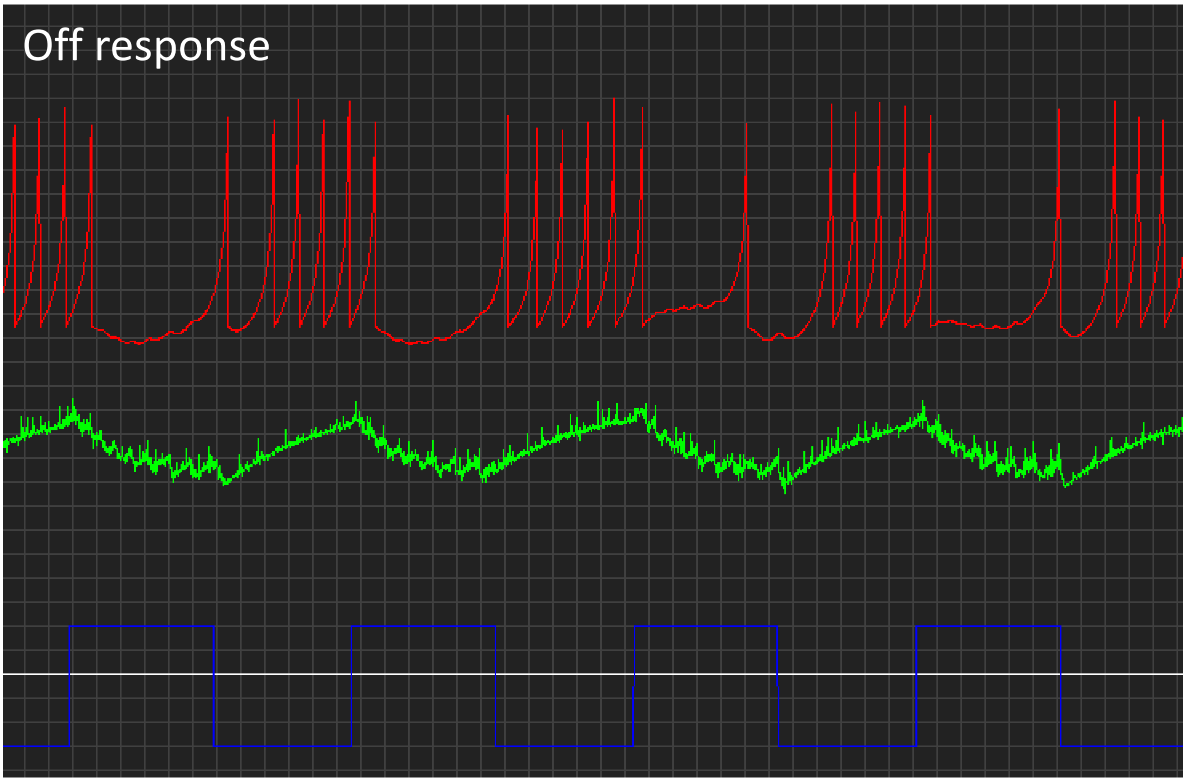
One "famous" coincidence detector is used in auditory systems, for comparing the signal between two ears. Imagine sound coming from your left. Because sound travels slowly (330 m/s) it will arrive at you left ear about a millisecond before it arrives at your right ear. Neurons in each ear's cochlea will spike as soon as the sound arrives and send that signal to the brainstem where the signals from the two ears are combined. Here there are bilateral neurons that receive inputs from both ears. These are the coincidence detectors. If the sound came from the left, the spike from the left ear comes in before the spike from the right ear and its associated postsynaptic current will decay rapidly, such that it does not overlap with the current triggered by a spike from the right ear. As a result, the central neuron will not spike. However, if the sound comes from e.g. straight ahead, it will reach both ears are the same time which in turn means that the left and right spike inputs to the central neuron will coincide and drive a postsynaptic spike. Such as central neuron therefore encodes the direction of sound. If we now slightly offset the speed by which the spikes travel, or implement delays of initial spike generation in either ear, we can build a neuron that is selective for any sound direction in the azimuth plane (horizontally around the head).

*This kind of logic expands into a multitude of computational operations. The above coincidence detector is perhaps the simplest of **logical operations**: an "AND" gate. This means that Input 1 AND Input 2 need to be "high" for the spike to be triggered. Another logical operation is an "OR" gate – Input 1 OR Input 2 need to be high. This is easily achieved by increasing the synaptic gain such that one neuron alone can drive the postsynaptic spike. There are many such logical operations (e.g. NOR, XOR, XNOR), which can be thought of a computational building blocks of electronics and neuronal networks alike. If you simulate many neurons in a computer, it is straightforward to implement any of these logical gates. We can* *try to set-up some of these using Spikeling. For example, how might you build an "inverse conditional" – i.e. a neuron that only fires if input 1 is high, unless input 2 is also high? How might you build a NOR gate (only fires when both Inputs 1 and 2 are "low"). How might you build an NAND gate (fires always except when both 1 and 2 are "high"). If you have more than 2 Spikelings at hand, try playing with these a bit. What kind of computations can you implement? (one useful reference for logical gates and their implementation and background can be found on Wikipeadia: https://en.wikipedia.org/wiki/Logical_connective).*

Note that in their "pure form", logical operations are Boolean ("True or False, no intermediate state is possible"). But neurons are noisy, so not all operations are equally "easy" to implement! (all are possible though).

Task 18. Using what you have learnt (and 2 Spikelings), can you build an "Off neuron"? This is a neuron that responds with spike(s) after the offset of a stimulus?

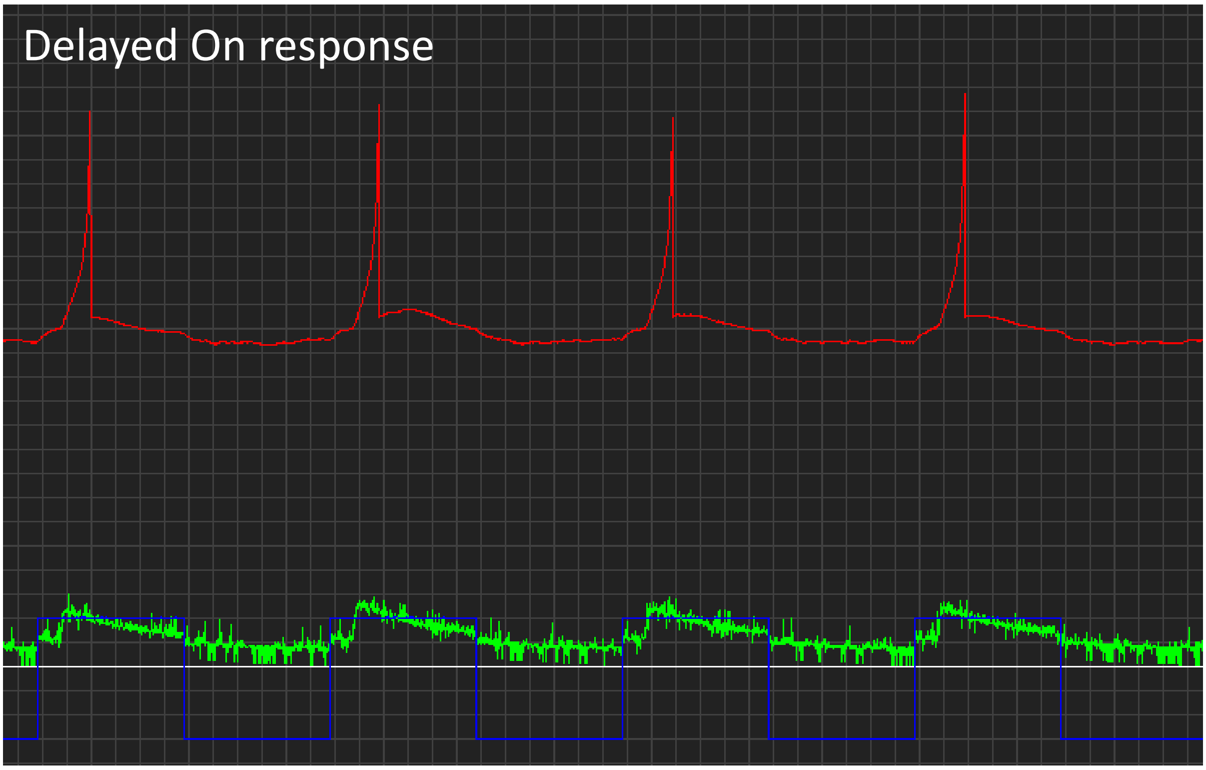
One way of doing this is by setting the postsynaptic neuron at a high resting potential such that it continuously generates spikes. Now take a presynaptic neuron that responds to the light in an "On fashion" and connect it to inhibit the continuously active cell. If you tune the inputs right, you should be able to generate something like the below:



Note that this is not a very "good" Off cell. Sometimes it spikes during the light, and more problematically, the timing of the 1st spike after the light switches off is very variable. This is because now we are not using the presence of an excitatory drive, but rather the absence of an inhibitory drive as signal to generate spikes. Using this setup, you will find it very difficult indeed to tune the off cell to become more transient. The problem is that the synaptic current of the inhibition is slow to decay so after the stimulus switches off the inhibition will take a while to settle back down to zero. This results in a slow upwards trend of the membrane voltage which, as we saw in the beginning, is a very ineffective way to trigger spikes. So to make a *"better" Off cell, we would need to implement something that make the membrane potential increase more rapidly after the inhibition turns off. There are many tricks that neurons use. The perhaps best one would be the implementation of a "**rebound spike**" behaviour, while another would be the use of a graded (non-spiking) network feeding into Spikeling (as we saw before, membrane voltage is much better at tracking fast inputs below spike-threshold). While Spikeling could principally be programmed to mimic either or both of these possibilities, we will not be covering them here.*

Task 19. Using what you have learnt (and 2 Spikelings), can you build an "delayed On neuron"? This is a neuron that responds with spike(s) after the onset of a stimulus, but with a delay?

One way of implementing this delayed on cell would be by use of an excitatory synapse that requires more than one presynaptic spike to trigger a postsynaptic spike as shown in the below. Note the two separate small depolarising current that precede each other by ~100 ms – these come from the presynaptic neuron responding to the light with 2 spikes in close succession. The second spike takes the postsynaptic neuron past threshold.



Task 20 (final!):

Using the delayed On-neuron circuit, plus an additional non-delayed On transient neuron, can you build an elementary motion detector? (Hint: This will require 3 neurons!)

Receptive fields

Blablabla