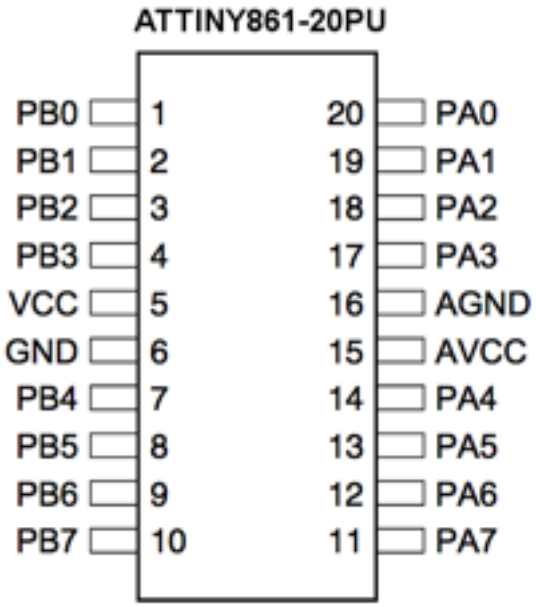
Tap-Tempo Clock, ATtiny861

By Harald Sabro. Version 1, 05.04.2014

Overview

The idea with this micro controller is making it possible to build a single unit/box for your pedalboard that allows you to set the tempo for all compatible effects at once. And there are a few bells and whistles included as well.

Pin for pin description



PB0 - **Half-speed clock output signal**, 0V-5V. This is the chip's main output signal and the one you typically want to pass on to other compatible micro controllers. Leave disconnected if not needed.

PB1 - **Full-speed clock output signal**, 0V-5V. If you want to interface with 3rd party units this is probably the clock signal they'd expect (though mind the voltage). Leave disconnected if not needed.

PB2 - [Tap input mode] LED indicating a tap tempo input is taking place. Pulled low when tempo counting is actively taking place. Leave disconnected if not needed. Will stay high in **clock input mode**.

PB3 - Unused. Leave disconnected.

PB4 - Crystal oscillator input. The chip expects an 8MHz crystal oscillator.

PB5 - Crystal oscillator input. The chip expects an 8MHz crystal oscillator.

PB6 - [Tap input mode] **Averaging mode**; average up to 10 tempo inputs to determine the best output tempo. Pull low to enable, disconnect to disable. Ignored in **clock input mode**.

PB7 - Unused. Leave disconnected.

PA0 - [Tap input mode] **Tap input switch**. Tap once to enter tempo counting, then tap again to set the tempo. Supports any tempo between 50ms and 10s. Leave disconnected if not needed. Ignored in **clock input mode**.

PA1 - [Clock input mode] Accepts a 5V **input clock signal** as basis for the output tempo. Leave disconnected if not needed. Ignored in **tap input mode**.

PA2 - **Sync input switch**. [Tap input mode] Tap once to realign the output clock signal. [Clock input mode] If the input clock signal is defined as **full-speed** (via **PA5** pin option), tap to extend the current **output clock pulse** by 50%. Leave disconnected if not needed.

PA3 - [Tap input mode] **Enable manual adjustments** (via **PA6**/**PA7** pins). Pull low to enable, disconnect to disable. Ignored in **clock input mode**.

PA4 - **Input selection**. Pull low to activate **clock input mode**, disconnect to activate **tap input mode**.

PA5 - [Clock input mode] Indicate **input clock signal is full-speed**. Pull low to enable, disconnect to disable. Ignored in **tap input mode**.

PA6 - [Tap input mode] **Manual adjustment** voltage controlled analog input. Leave at AVCC/2 for no adjustment; shift towards GND to decrease tempo, or shift towards AVCC to increase tempo. Leave disconnected if not needed. Ignored in **clock input mode**.

PA7 - [Tap input mode] **Manual adjustment range** voltage controlled analog input. Control the intensity of the **manual adjustment input** (**PA6**). Towards GND the adjustment input becomes fine grained, and towards AVCC the adjustment input becomes very intense. Leave disconnected if not using averaging, or use an internal trimpot to set a permanent default setting. Ignored in **clock input mode**.

Selecting input mode

The chip needs some form of input to be able to produce an **output clock signal** at the desired tempo/frequency (in the absence of any input or other modifying conditions the chip will boot up outputting a 0.5Hz clock signal). There are two mutually exclusive alternatives; **tap input mode** or **clock input mode**, both of which will be described in more details further down.

The **PA4** pin controls which of the **input modes** are currently active. To activate **clock input mode** this pin must be kept at ground potential, and to activate **tap input mode** the pin can be left disconnected (where an internal pull-up resistor will raise it to VCC potential). This means the **input selection** can be controlled by a basic SPST off-on switch, and it also implies that, should you only be interested in the **tap input mode** this pin can be left entirely disconnected.

Tap input mode

This is the **default mode** and is based around the **tap input switch** pin (**PA0**). The **tap input switch** pin should be connected to a single pole momentary/non-latching switch suitable for stepping on, with the other side of this switch connected to ground.

The pin uses debouncing logic to detect whenever a stable connection takes place, something like 10 milliseconds of solid connection to ground, which filters out most noise and switch bouncing. The switch “tap” is registered on the downward edge, i.e. as the switch is pressed rather than released, after which it can be safely released. In other words, it is only necessary to give the switch a quick push each time it needs to be operated.

The first time a tap is registered the chip enters **tempo counting state**, at which point it starts counting milliseconds waiting for a second tap to indicate the final tempo. If another tap is registered within the maximum allowed timeframe (10 seconds) the new tempo is immediately accepted and the **output clock signal** changed accordingly (not taking into account **manual adjustment** modifiers, see below). If no second tap is registered within the maximum timeframe, or the **input mode** is changed to **clock input mode**, the **tempo counting state** is abandoned and the **output clock signal** is left unaffected.

The **tap input LED indicator** pin (**PB2**) is pulled and kept low as long as the chip is in the **tempo counting state** (only works in **tap input mode**). This can be used to light a LED making it easy to see whether the chip is actively counting tempo or not.

The **sync input switch** pin (**PA2**), should either be left disconnected if not needed, or, like the **tap input switch** pin, be connected to a single pole momentary/non-latching switch. This switch works in both **input modes**, but with slightly different results. In **tap input mode** this switch can be stepped on once to instantly realign the **output clock signal** without entering the **tempo counting state** (though it will abandon any **tempo counting state** that is already in progress) and without affecting the tempo itself.
Imagine a given tap tempo with such a small error as to be indistinguishable in a short timespan, but that will have drifted a perceptible amount after a greater length of time. Instead of starting the tempo input process from scratch, a single tap on this switch on the beat will bring the **output clock signal** right back to where you want it.

TEMPO ADJUSTMENT FEATURE

This feature is only available in **tap input mode** and is enabled by pulling and keeping the **manual adjustment enable** pin (**PA3**) low. To disable, or if not needed, this pin can be left disconnected. I.e. it’s a perfect candidate for an SPST off-on toggle switch.

When enabled two new pins come into play; the **manual adjustment input** pin (**PA6**) and the **manual adjustment range input** pin (**PA7**). Both pins are voltage controlled analog inputs that accepts a voltage somewhere between ground and AVCC.

Assuming that a linear potentiometer is connected to the **manual adjustment input** pin (**PA6**) the neutral position is at AVCC/2 or 12 o’clock (keep in mind that most potentiometers have quite wide tolerance). The closer this pin is pulled towards ground the more the **output clock signal** will be slowed down. And the opposite direction, pulling the pin closer to AVCC, the **output clock signal** will be increasingly sped up.
Note that this does not affect the base tap tempo, and disabling this feature will result in the **output clock signal** reverting to match the base tempo.

The **manual adjustment range input** pin (**PA7**) is a straight forward sensitivity control that increases the intensity of the **manual adjustment input** as the pin is pulled from ground towards AVCC (in approx. 10 distinct steps).
At the lowest setting (pin at ground potential) the **manual adjustment control** (**PA6**) will be able to modulate the **output clock signal** by approx. 128 milliseconds in either direction (slower/faster).
At the highest setting (pin at AVCC potential) the **manual adjustment control** (**PA6**) will be able to modulate the **output clock signal** by as much as 1.28 seconds in either direction (slower/faster).

If the **manual adjustment input** pin is going to be used alone the **range** pin should be permanently set at the preferred setting, typically with an internal trimpot. If left disconnected it will be pulled to AVCC by the internal pull-up resistor and consequently be at “full range”.

Now, using a potentiometer for the **manual adjustment input** is perfectly fine, but I imagine feeding this pin some other form of control signal could be interesting too! (Maybe an envelope signal or an LFO signal?) Make it switchable for maximum flexibility.

TAP-TEMPO INPUT AVERAGING FEATURE

Typically as soon as a new input tempo is input the previous one is discarded, but enabling this feature (only available in **tap input mode**) allows up to 10 tempo counts to be averaged for a, hopefully, more accurate final tempo.

The feature is enabled by pulling and keeping the **averaging enabled** pin (**PB6**) low. To disable the feature, or if it’s not needed, the pin is left disconnected.

When enabled the chip will start saving each new tempo up to a maximum of 10, whereupon the oldest tempo will be overwritten etc. The **output clock signal** is set to the average (though this tempo may still be affected by a **manual tempo adjustment** should that feature also be enabled).

To discard all currently stored tempos the feature need only be disabled for a brief moment (via **PB6**).

Clock input mode

This mode is based on receiving and synchronizing against an **external clock signal** rather than relying on the **tap input switch**. The clock signal is continually read from the **input clock signal** pin (**PA1**) and the **output clock signal** is modulated accordingly.

In this mode the **manual adjustment**- and **tempo averaging** features are disabled. Since the chip continually re-aligns with the incoming clock pulses trying to do something like extend the output clock pulses will only result in the next clock pulse interrupting the previous one before it has time to complete.

The chip natively runs on a “**half-speed**” clock frequency; more on that below, but it is worth noting that this is the expected **input clock signal** by default.

If you want to synchronize based on a “**full-speed**” clock signal, say from a 3rd party unit, click track etc., the “**input clock signal is full-speed**” feature should be enabled by pulling and keeping the **full-speed cock input** pin (**PA5**) low. If running off a “**half-speed**” clock, or if this feature is not needed at all, this pin is left disconnected.

When configured to accept a “**full-speed**“ clock signal the **sync input switch** pin (**PA2**) becomes an option. Because a “**full-speed**“ **input clock signal** will result in a “**half-speed**” **output clock signal**, there will be two **input clock pulses** for each one **output clock pulse**. A single tap on the **sync input switch** will result in the next **output clock pulse** being extended by 50%, thereby switching the **output clock signal** from aligning with one **input clock pulse** to aligning with the other **input clock pulse** instead.

Output clock signals

The chip outputs both a native “**half-speed**” clock signal (**PB0**) and a “**full-speed**” output signal (**PB1**), where the full-speed signal is just a doubling of the half-speed one.

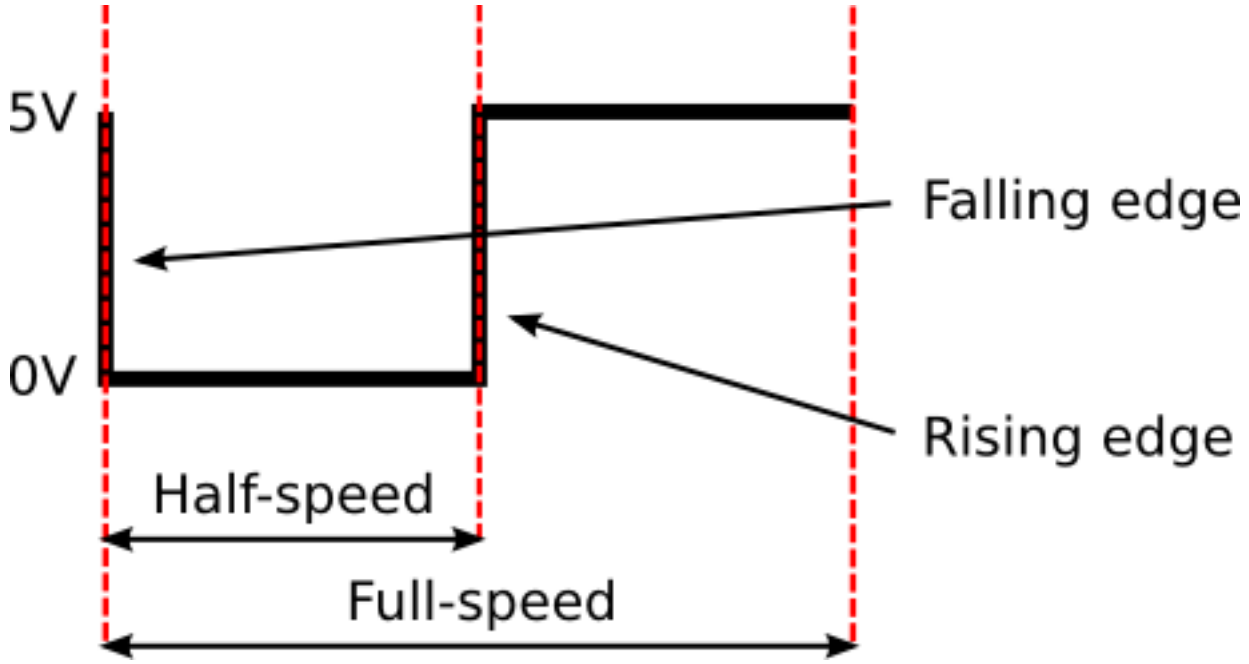
If driving other compatible chips, like the **LFO v2**, and probably other chips in the future, they will be expecting the “**half-speed**” clock signal. Read more about the reason for this below.

But a “**full-speed**” signal is also included for passing a clock signal to 3rd party devices etc. that probably expects a proper signal. Though do mind the amplitude (5V) and make sure the receiving device can handle it.

There’s an interesting quirk here in the way a changing input clock frequency impacts the output clock signals differently. Because the chip internally runs on “**half-speed**” frequencies, the corresponding half-speed output signal is instantly matched, while the full-speed output signal continues at the previous frequency until the new one has been fully established. The reason for this will (hopefully) be evident after reading the next section.

Half-speed and full-speed clock signals

There’s been a lot of talk about half- and full-speed clock signals, but no explanation as to what it really is, so let’s try to do something about that.



Defining the tempo when running in **tap input mode** requires two taps; one to initiate a millisecond counter, and one to stop the counting and set the tempo based on this count. The result is a clock output signal of a given frequency relating to the input tempo, with it’s signal edges aligned in time to the taps.

Now, let’s assume the output clock signal is “full-speed”; during one clock pulse both the falling- and rising edge have to be transmitted, with the rising edge coming exactly half-way through the complete pulse.
Considering the arbitrary distance between a user’s “start”- and “stop” taps, there’s no way of knowing ahead of time when we’ve reached the half-way point between these to inputs. This again

means we cannot propagate a new output clock signal until we know it's frequency; i.e. the final user defined tempo. The result is a propagation latency of one full clock cycle from when a user inputs a tempo to the time a second chip receiving the output clock signal has adjusted to this same tempo. The chip allows a clock signal of as much as 10 seconds between each pulse. No good!

Now, let's assume we want to output a "**half-speed**" signal. This time we consider the user's "start"- and "stop" taps as the falling- and rising edge of a corresponding output clock signal. As soon as the first tap is registered, a falling edge is produced on the output clock signal, and likewise, as soon as the second tap is registered, a corresponding rising edge is produced on the output clock signal. We've just "mirrored" the user's tap-tempo on the output clock signal in real-time, though we've only really presented half the clock cycle. Now that the final tempo is known the chip can "keep quiet" for the second half of the clock cycle, before starting over again with a falling edge for the next output clock pulse. Etc.

As long as the receiving chip treats the clock signal this way as well (counting tempo between falling- and rising edges), there's no propagation latency at all. Theoretically n-number of chips can be chained together, clock outputs to clock inputs, and a single user-input tap-tempo will instantly set the new tempo on all chips. Much better!

And then really, the reason for doing it this way is because that's how I chose to implement it in the ATtiny85 LFO chip, not really considering clock signals or frequencies at all beyond a practical level ;)

Harald