Snaiks

Signals and Systems from KiCad to C March 2017 This page is a successor of the [[Snaiks-Study]] ---- ===== Introduction ===== {{ ::snaiks-logo.png?nolink&200 }} Snaiks provides a tool chain to get from a signal plan, drawn in KiCads schematic editor to a generated standard C** code, runnable on **any platform**. This tool chain consists of following parts: * **Snaiks KiCad Library**, providing all the symbols * **Snaiks Compiler**, generating C++ code out of KiCads netlist * **Snaiks C++ Library**, defining the functionality of the KiCad symbols It's purpose is to create complex systems by drawing them in KiCad's schematic editor and generate out of the netlist a working C++ code, which also compiles for micro controllers without dynamic memory allocation. It can be used to implement PLCs or digital signal processing like filtering. ===== Goals ===== * Generate beautiful C++ code from a KiCad schematic * Compiles without dynamic memory allocation (embedded, safety) * Read and write system states during runtime (e.g. with a simple terminal) * Simple custom system creation (KiCad component editor + sub-class implementation) * Hierarchical design (sub-systems) * full documentation within the schematic ===== Status ===== ==== Implemented ==== * A set of basic blocks in KiCad and C++ * Snaiks Compiler, in alpha state * auto arrangement of block-execution to provide transparent behavior ==== Open Tasks ==== * communication with the system in runtime * unit tests of all Snaiks Objects ===== Mini-Demo ===== [[https://gitlab.com/KarlZeilhofer/snaiks-demo-2 This demo project on GitLab.com]] shows the usage of snaiks in a simple console application, a Qt app with GUI and a realtime-application running on the RevolutionPi ===== Installer ===== TODO: make an installer repo, which clones all the sub-repos, compiles and installs the snaiks compiler, installs kicad and a demo project.**

```
# using git clone --recursive to check out all submodules
git clone --recursive https://github.com/chaconinc/MainProject
```

===== Source Code ===== * https://gitlab.com/KarlZeilhofer/snaiks-cpp-lib * https://gitlab.com/KarlZeilhofer/snaiks-kicad-lib * https://gitlab.com/KarlZeilhofer/snaiks-compiler ===== Library Presentation ===== * https://gitlab.com/KarlZeilhofer/snaiks-kicad-lib-presentation *

Presentation as PDF

(perhaps outdated) ===== Blue Prints ===== ==== Properties ==== A Snaiks component can have properties. For example: * monoflop period * schmitt trigger limits * saturation limits * corner frequency or filter-type of a digital filter * filter coefficients * gain value * value of constants A property consists of * a value * a name * a persistent initial value * a setter method * a getter method * a method to store a changed value into the persistent memory ==== Info-System ==== A system generated by Snaiks should be fully discoverable and manipulatable during runtime. === Use cases === * change filter characteristics * change regulator parameters * adjust offset or gain * change system constants * change enable/disable flags * reset a component or the whole system * start/stop recording === Needed Features === * list inputs and outputs of an object * list properties of an object * change property values permanently ==== Any-Type Inputs/Outputs ==== Perhaps it would be useful, that not all inputs must have the same type. For example a mute gate, where the enable is bool and the signal is double. Pros: * more flexible systems Cons: * every pin must have a type specified in KiCad (could be done with net-annotators, similar to PWR_FLAG). * we cannot use a simple template-interface class any more === Proposal === * in cases, where this is really needed, a specific C class could be implemented * mixture of numbers and bool shouldn't be any problem english, software, signals, kicad, snaiks, technical