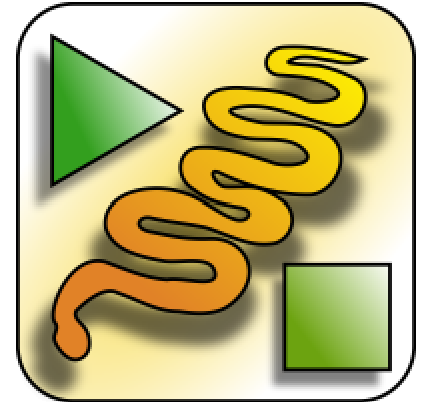
Snaiks Study

Signals and Systems from KiCad to Cpp



7.4.2016
For Updates please see Snaiks

Introduction

Signals and Systems is here a collection of Cpp classes on one hand, and a collection of corresponding KiCad components on the other hand.

It's purpose is to create complex systems by drawing them in KiCad's schematic editor and generate out of the netlist a working Cpp code, which also compiles for micro controllers without dynamic memory allocation.

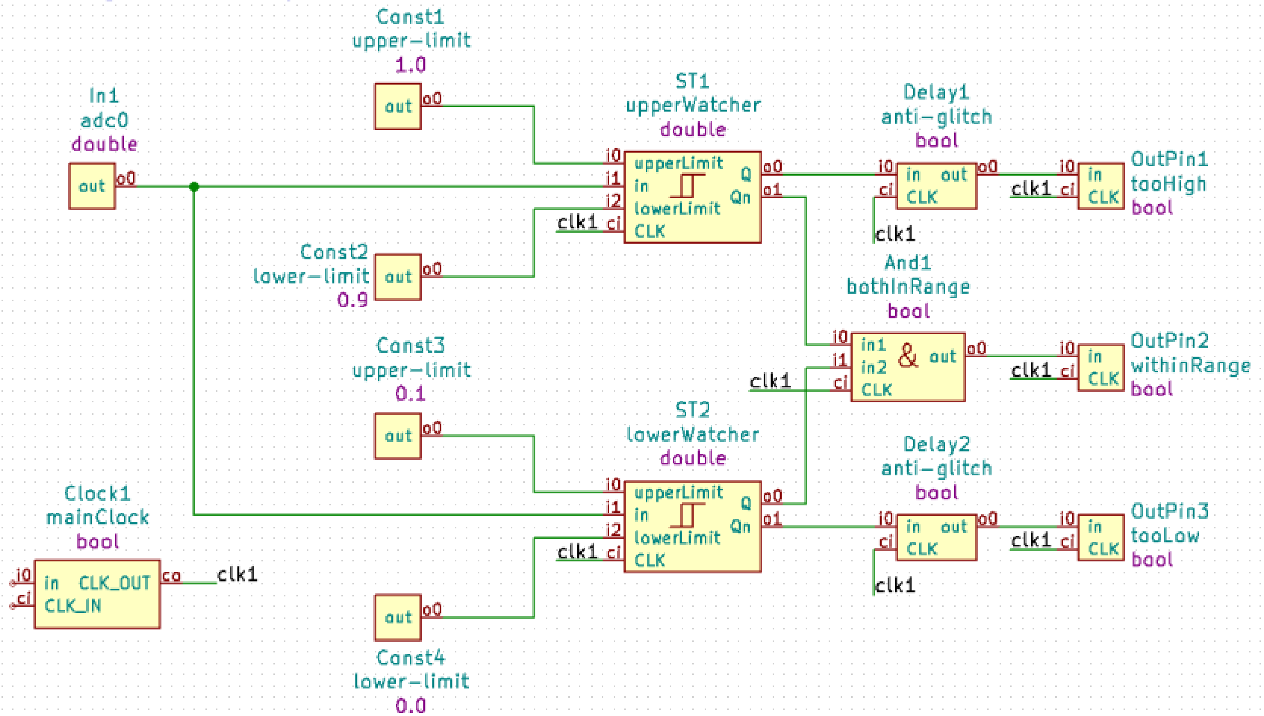
It can be used to implement PLCs or digital signal processing like filtering.

The Cpp classes are based heavily on templates, so most of the components can be used either for floating point or for integer calculations.

Goals

- Generate beautiful Cpp code from a KiCad schematic
- Compiles without dynamic memory allocation (embedded, savety)
- 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

Mini-Demo




```
SnsConstant<double> Const1(1.0);
SnsConstant<double> Const2(0.9);
SnsConstant<double> Const3(0.1);
SnsConstant<double> Const4(0.0);

SnsInput<double> In1(0.0);

SnsSchmittTrigger<double> ST1;
SnsSchmittTrigger<double> ST2;

SnsLogicAnd<bool> And1;

SnsOutputCout<bool> Out1;
SnsOutputCout<bool> Out2;
SnsOutputCout<bool> Out3;

SnsDelay<bool> Delay1;
SnsDelay<bool> Delay2;

SnsClockMaster Clock1(8);
```

```
void sns_build()
{
    ST1.setSource(0, Const1.getOuput(0));
    ST1.setSource(1, In1.getOuput(0));
    ST1.setSource(2, Const2.getOuput(0));

    ST2.setSource(0, Const3.getOuput(0));
    ST2.setSource(1, In1.getOuput(0));
    ST2.setSource(2, Const4.getOuput(0));

    Delay1.setSource(0, ST1.getOuput(0));
    Delay2.setSource(0, ST2.getOuput(1));

    And1.setSource(0, ST1.getOuput(1));
    And1.setSource(1, ST2.getOuput(0));

    Out1.setSource(0, Delay1.getOuput(0));
    Out2.setSource(0, And1.getOuput(0));
    Out3.setSource(0, Delay2.getOuput(0));

    Clock1.addSlave((SnsSystem*)&ST1);
    Clock1.addSlave((SnsSystem*)&ST2);
    Clock1.addSlave((SnsSystem*)&And1);
    Clock1.addSlave((SnsSystem*)&Out1);
    Clock1.addSlave((SnsSystem*)&Out2);
    Clock1.addSlave((SnsSystem*)&Out3);
    Clock1.addSlave((SnsSystem*)&Delay1);
    Clock1.addSlave((SnsSystem*)&Delay2);
}
```

```
int main()
{
    sns_build();

    for(double in=-0.5; in<=1.5; in+=0.077777)
    {
        printf("In1=%+1.2f \t", in);
        In1.setValue(in);
        Clock1.sample();
        Clock1.process();
        std::cout << std::endl;
    }
    return 0;
}
```

```
// output (now without glitches at the transitions, see test1):
/*
In1=-0.50   0 0 0
In1=-0.42   0 0 1
In1=-0.34   0 0 1
In1=-0.27   0 0 1
In1=-0.19   0 0 1
In1=-0.11   0 0 1
In1=-0.03   0 0 1
In1=+0.04   0 0 1
In1=+0.12   0 0 1
In1=+0.20   0 0 1
In1=+0.28   0 1 0
In1=+0.36   0 1 0
In1=+0.43   0 1 0
In1=+0.51   0 1 0
In1=+0.59   0 1 0
In1=+0.67   0 1 0
In1=+0.74   0 1 0
In1=+0.82   0 1 0
In1=+0.90   0 1 0
In1=+0.98   0 1 0
In1=+1.06   0 1 0
In1=+1.13   0 1 0
In1=+1.21   1 0 0
In1=+1.29   1 0 0
In1=+1.37   1 0 0
In1=+1.44   1 0 0
*/
```

Source Code

- https://gitlab.com/KarlZeilhofer/Snaiks-cpp-lib
- https://gitlab.com/KarlZeilhofer/Snaiks-kicad-lib
- https://gitlab.com/KarlZeilhofer/Snaiks-manual-tests

Blue Prints

Sampled vs. Transparent Systems

The Problem

For some systems it would be handy, if they are transparent from input to output.
This means, that they do not consume a whole clock cycle. An example could be a simple And-gate. Perhaps one do not want this gate to introduce an extra clock-cycle, until the combined signal is visible on the output.
The drawback of this method is, that the update-order of all the systems is very important. If the user is aware of this problem, he can shorten latencies of complex systems.

In Snaiks example test1 we had a undesired behaveour, because the And-gate befor the middle output indroduced a delay of one clock. In transitions from „within range“ to „too high“ and to „too low“ undesired states were produced, were either 2 outputs were true at the same time, or where no output was true for one clock cycle.
This made the antiGlitchDelays necessary.

The Proposal

Every System has a flag **transparent** which is by default false.
If it is true and sample() is called, update() is called from the sample() function:

```
class MySystem
{
    ...
    bool transparent;
    bool sampled=false;
    ...
    void sample()
    {
        sInput = *input;
        sampled = true;
        if(transparent)
        {
            update();
        }
    }

    void update()
    {
        if(sampled)  // avoid double update
        {
            sampled = false;
            output = ...
        }
    }
    ...
}
```

This also makes the flag sampled necessary. Becaus the normal clock master calls every sample() and then every update(). A second update-run shouldn't change anything, but consumes perhaps unnecessay processing time.

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

Hierarchical Systems

It should be possible to combine a set of systems to a sub-system, where new inputs and outputs are defined.

KiCads hierarchical schematic structure could be used out of the box, but in Cpp we do not see anything from this. Similar to the KiCad PCB layout, which also doesn't know anything about a hierarchical sheets. Altium Designer has the so called rooms, which group and synchronize footprint arrangement and traces between multiple instances of one hierarchical sheet.

For the systems we should make something similar to the properties, which live in a SnsPropertyContainer. We should make a SnsSystemContainer, which provides on one hand memory for the different objects of a sub-system and on the other hand new inputs and outputs.

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, such as the SnsHybrid or SnsNumeric.

Proposal

- in cases, where this is really needed, a specific Cpp class could be implemented
- mixture of numbers and bool shouldn't be any problem

english, software, signals, kicad, snaiks, technical

From:
http://www.zeilhofer.co.at/wiki/ - **Verschiedenste Artikel von Karl Zeilhofer**

Permanent link:
http://www.zeilhofer.co.at/wiki/doku.php?id=snaiks-study

Last update: **2024/09/19 12:16**