Signals and Systems

7.4.2016

Introduction

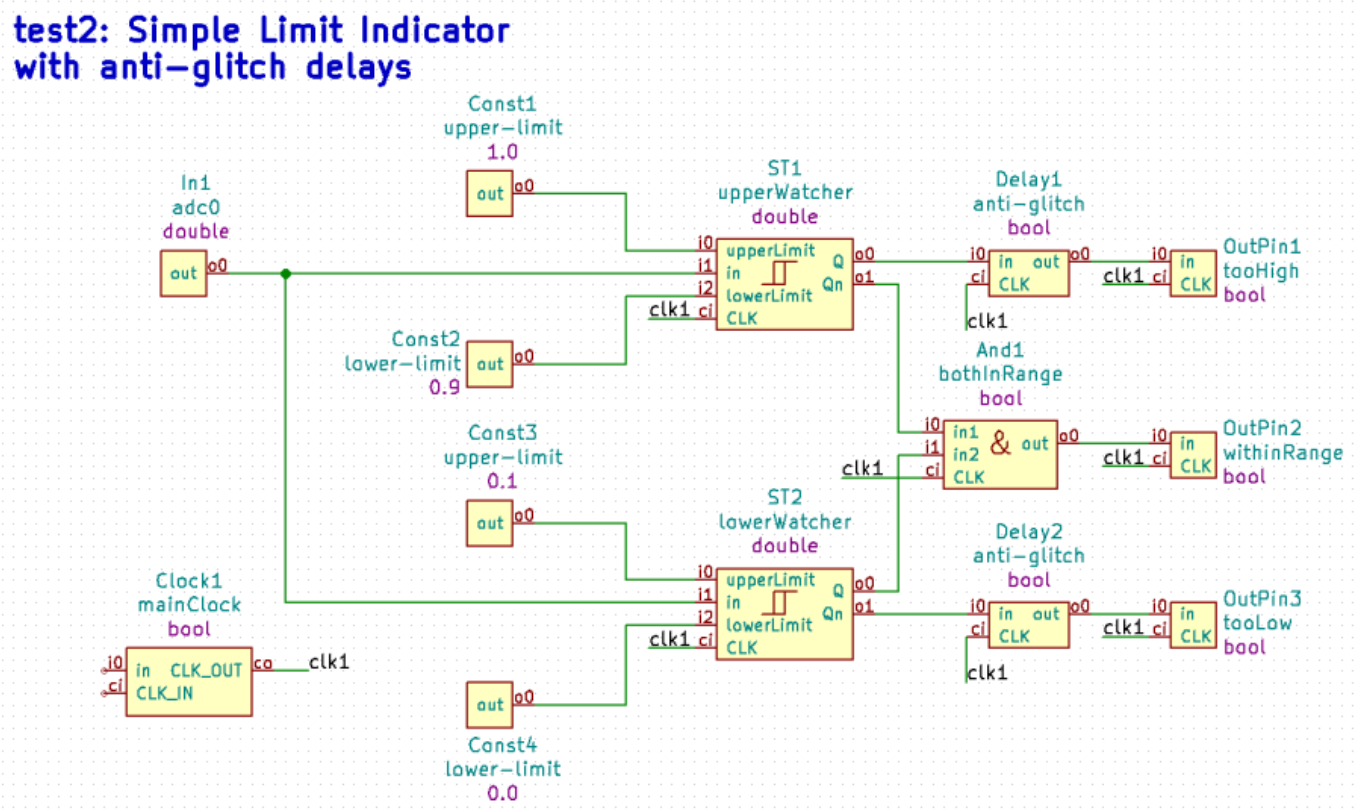
Signals and Systems is here a collection of C++ classes on one hand, and a collection of corresponding KiCad components on the other hand.

It's purpose is to create complex systems by drawing them in KiCad's schematic editor and generate out of the netlist a working 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.

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

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