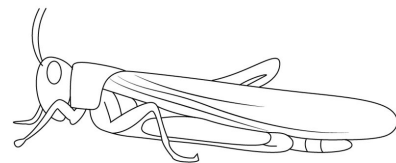# Locust development tutorial workshop (How to add a new Locust Generator)
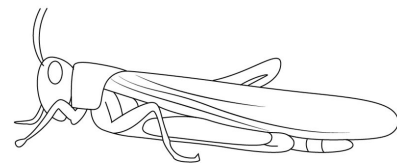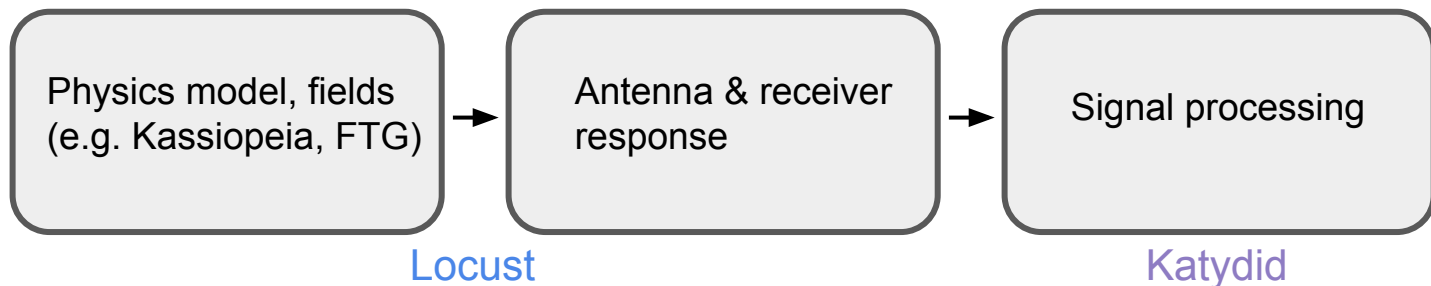
P. L. Slocum
Jan. 28, 2021

Contents:

- This is a brief introduction to developing in Locust, in p8compute.

- For running previously-developed Locust with examples, please see the tutorial at
  https://github.com/project8/locust-tutorial/blob/master/LocustTutorial.pdf

- We will start with a description of the workflow.

  - Typical CRES-like workflow.

  - Today's tutorial workflow (a square wave signal, without CRES).

- Finally, we will write a new Generator and process its output.

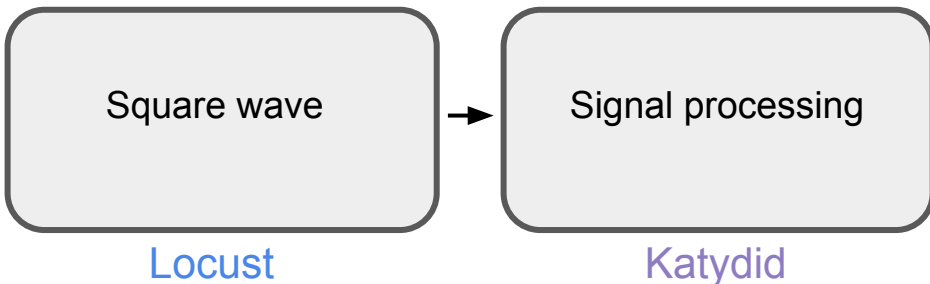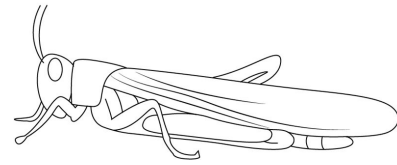- Technical questions can be posted in Slack in the new #locust channel.

In this tutorial:

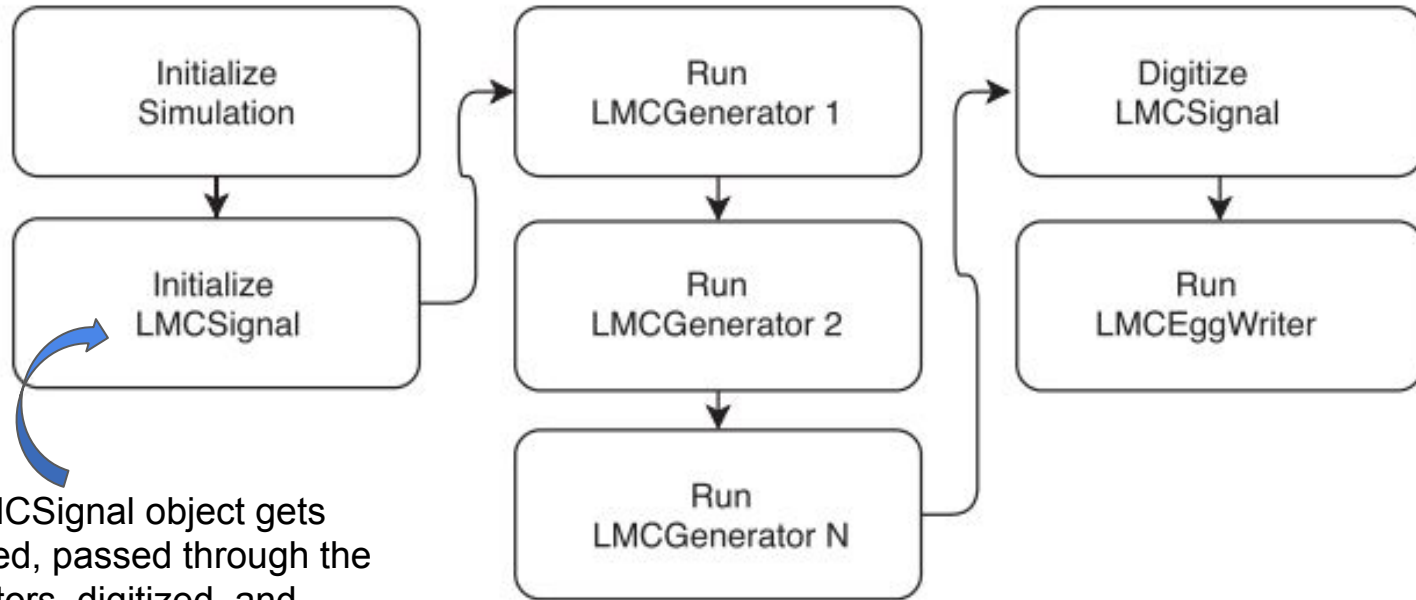Typical workflow in Locust-Katydid:

```
┌─────────────────────┐     ┌─────────────────────┐     ┌─────────────────────┐
│ Physics model, fields│ →  │ Antenna & receiver   │ →  │ Signal processing    │
│ (e.g. Kassiopeia, FTG)│     │ response             │     │                      │
└─────────────────────┘     └─────────────────────┘     └─────────────────────┘
                                    Locust                        Katydid
```

Today's tutorial:

```
          ┌─────────────────────┐     ┌─────────────────────┐
          │ Square wave          │ →  │ Signal processing    │
          └─────────────────────┘     └─────────────────────┘
                  Locust                      Katydid
```
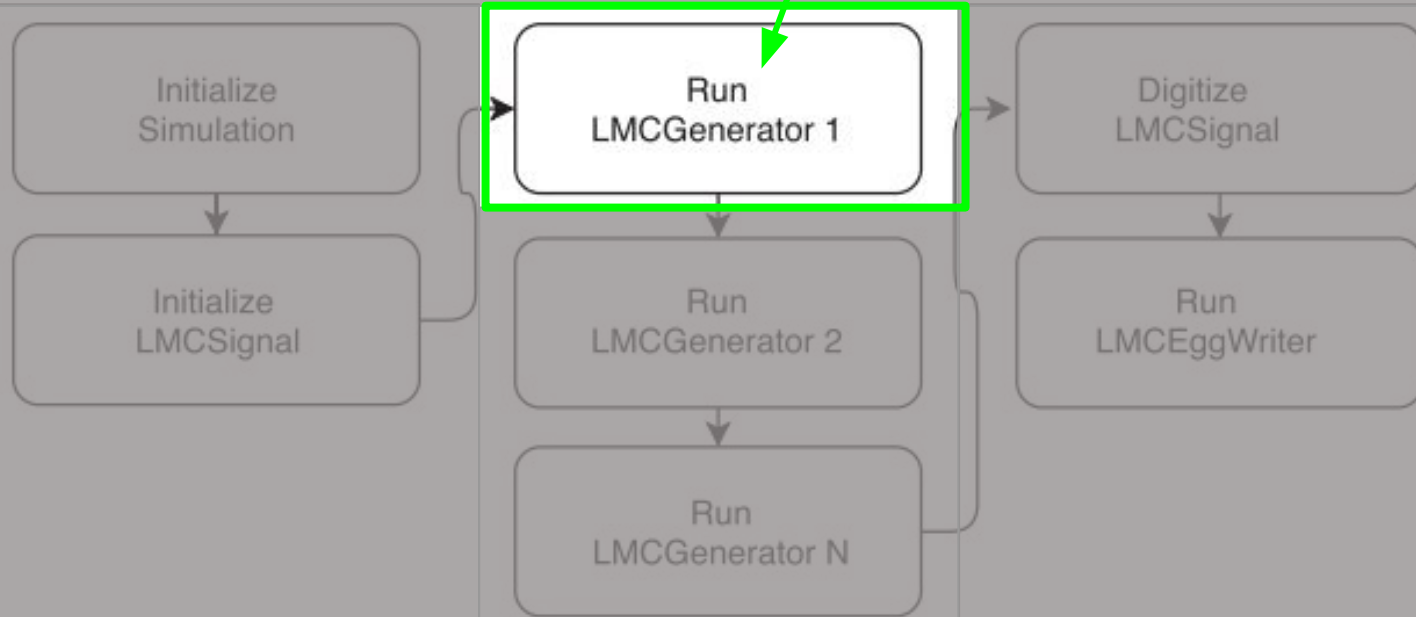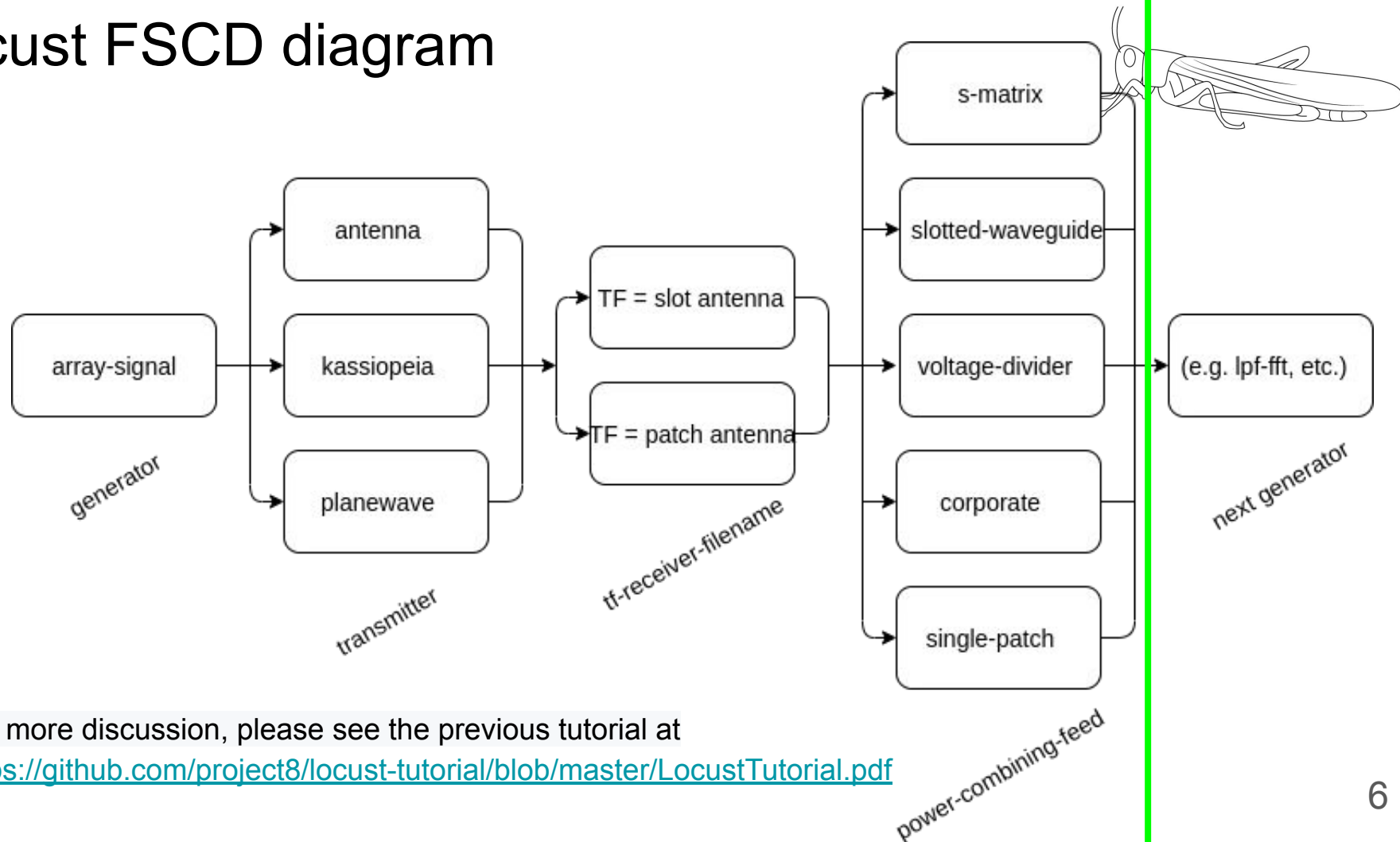
3

Locust work flow diagram

The LMCSignal object gets initialized, passed through the generators, digitized, and written to an egg file.

Locust work flow diagram

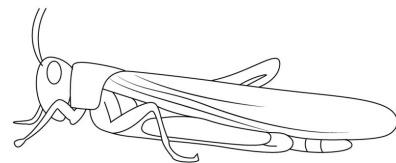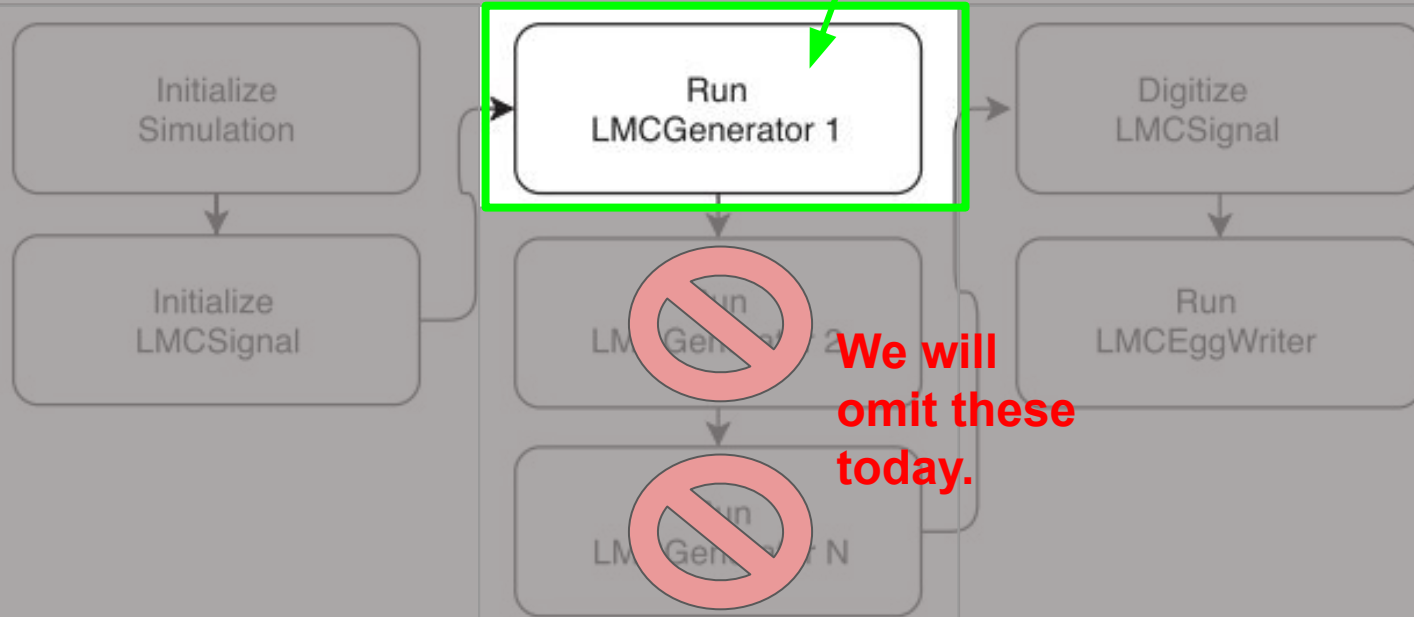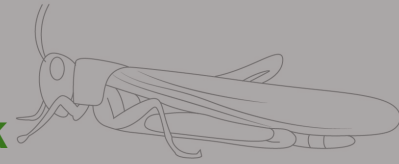**Typical location for physics interface and antenna calculations.**

Initialize Simulation	Run LMCGenerator 1	Digitize LMCSignal
Initialize LMCSignal	Run LMCGenerator 2	Run LMCEggWriter
	Run LMCGenerator N	

Locust FSCD diagram

For more discussion, please see the previous tutorial at
https://github.com/project8/locust-tutorial/blob/master/LocustTutorial.pdf

6

Locust work flow diagram

For today we will start from a blank template, here.

Initialize Simulation

Initialize LMCSignal

Run LMCGenerator 1

Run LMCGenerator 2

Run LMCGenerator N

We will omit these today.

Digitize LMCSignal

Run LMCEggWriter

First, some steps for setting up

Before we start the examples:

1. Install docker as in https://docs.docker.com/get-docker.
2. sudo docker pull project8/p8compute (sudo is unnecessary on Macs)
3. sudo docker pull project8/p8compute-jupyter (sudo is unnecessary on Macs)
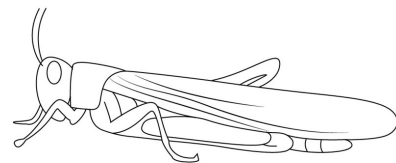4. Create a directory in your home directory, called ~/p8tutorial:

   mkdir ~/p8tutorial

5. In the ~/p8tutorial directory, clone (or pull) the locust-tutorial repo:

   cd ~/p8tutorial
   git clone git@github.com:project8/locust-tutorial

First, some steps for setting up (cont.)

7. Now clone Locust into ~/p8tutorial:

   ```
   cd ~/p8tutorial
   git clone git@github.com:project8/locust_mc
   cd locust_mc
   git checkout feature/tutorial
   git submodule update --init --recursive
   ```
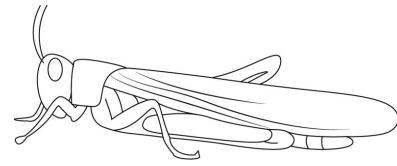
8. Start p8compute-jupyter and leave it open for the duration of the workshop:

   (all one line): ```docker run -p 8888:8888 -v ~/p8tutorial/locust-tutorial:/work -v ~/p8tutorial/locust_mc:/locust_mc project8/p8compute-jupyter```

9. Open a browser tab using one of the links provided in the resulting terminal output.

Start the p8compute container, compile Locust

- Start the container mount the Locust and work directories:
  (all one line): docker run -it -v ~/p8tutorial/locust_mc:/locust_mc -v ~/p8tutorial/locust-tutorial:/work project8/p8compute:latest /bin/bash
- Compile Locust
  source /usr/local/p8/compute/v0.10.1/setup.sh
  cd /locust_mc
  mkdir cbuild
  cd cbuild
  mkdir output/   (for https://github.com/project8/locust_mc/issues/177)
  cmake ../
  make install
  Or, if compiling with Kassiopeia,
 cmake -Dlocust_mc_BUILD_WITH_KASSIOPEIA=ON ../

Create a new square wave signal Generator
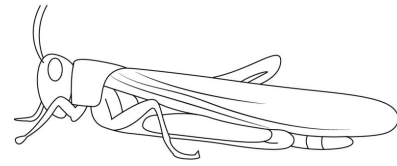
- Outside the container (if done from inside then change file permissions with chmod a+w):

  Skip for today Jan. 28

  cd ~/p8tutorial/locust_mc/Source/Generators
  cp LMCTemplateGenerator.cc LMCSquareWaveSignalGenerator.cc
  cp LMCTemplateGenerator.hh LMCSquareWaveSignalGenerator.hh

- Outside the container, in your favorite text editor, IDE, or both, open 6 files:

  Source/Generators/LMCSquareWaveSignalGenerator.cc,
  Source/Generators/LMCSquareWaveSignalGenerator.hh
  Source/Core/LMCRunLengthCalculator.cc,
  Source/Core/LMCRunLengthCalculator.hh
  Source/Core/LMCVisitor.hh
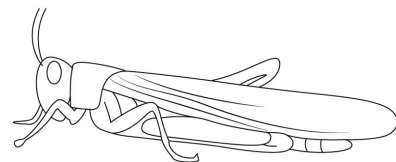 Source/CMakeLists.txt
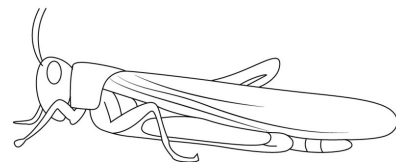
Edit the 6 files:

- In Source/Core/LMCRunLengthCalculator.cc uncomment these lines:

```
/*
void RunLengthCalculator::Visit( const SquareWaveSignalGenerator* )
{
 // nothing to see here, move along, please
 return;
}
*/
```

- In Source/Core/LMCRunLengthCalculator.hh uncomment this line:

```
// void Visit( const SquareWaveSignalGenerator* );
```

Edit the 6 files, cont.

- In Source/Core/LMCVisitor.hh uncomment this line:
  // class SquareWaveSignalGenerator;

- In Source/CMakeLists.txt, under set(LOCUST_MC_HEADER_FILES)
  uncomment this line:
  # Generators/LMCSquareWaveSignalGenerator.hh

- In Source/CMakeLists.txt, under set(LOCUST_MC_SOURCE_FILES)
  uncomment this line:
  # Generators/LMCSquareWaveSignalGenerator.cc
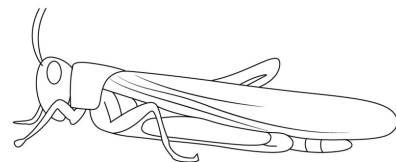
Now, edit the new generator:

- In LMCSquareWaveSignalGenerator.hh:

  Replace [NAME] with SQUAREWAVESIGNAL (3 times).
  Replace [name] with SquareWaveSignal (6 times).
  Replace [domain] with Time (1 time, in comment).
  Replace config-name with square-wave (2 times).
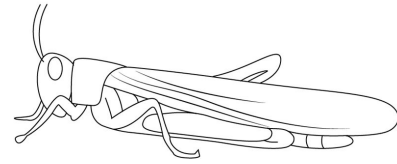
- In LMCSquareWaveSignalGenerator.cc:

  Replace [name] with SquareWaveSignal (18 times).
  Replace [domain] with Time (1 time).
  Replace [config-name] with square-wave (1 time).
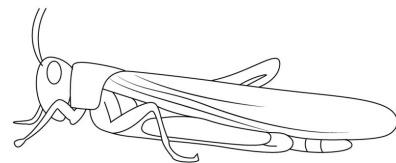 Edit the name and date in the comment.

Recompile Locust in the container

- This is a short slide:
  cd /locust_mc/cbuild
  cmake ../
  make install

Now we need a config file

- Inside the container (or from outside container):

  cd /locust_mc/cbuild/config
  cp LocustBlankTemplate.json /work/LocustSquareWaveTemplate.json
  (If done from inside the container, you may need to change write permission for editing):
  chmod a+w /work/LocustSquareWaveTemplate.json

- Outside the container in text editor, open the new
  ~/p8tutorial/locust-tutorial/LocustSquareWaveTemplate.json, add your new generator to the
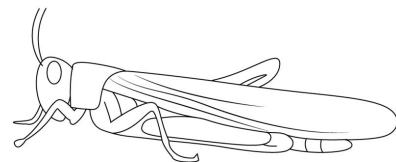 list of "generators", and save it:

  "generators":
  [
      "square-wave",
      "digitizer"
  ],

Test: Run the new generator

- In the container, run your locally-cloned, and now expanded, Locust:

 (all one line): `/locust_mc/cbuild/bin/LocustSim config=/work/LocustSquareWaveTemplate.json`

  Confirm that the output egg file is written to /locust_mc/cbuild/output/locust_mc.egg . (It will contain all zeroes.)

  `ls -l /locust_mc/cbuild/output/locust_mc.egg`

- Process the egg file with Katydid in the container, like this:

  `Katydid -c config/katydid.json`

# Now, outside container, edit the Generator to produce a square wave
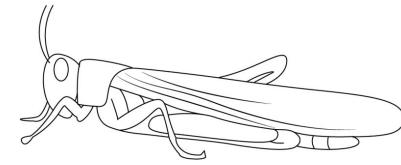
- In LMCSquareWaveSignalGenerator.cc, uncomment these lines in function DoGenerateTime(Signal* aSignal):

```
/*
double RF_freq = 50.e6; // Hz
double amplitude = 1.e-5; // volts


        for( unsigned index = 0; index < aSignal->TimeSize(); ++index )
        {
            int nRFHalfCycles = int(index*(1./(fAcquisitionRate*1.e6)*(2.0*RF_freq)));
//            int nRFHalfCycles = int(index*(1./(fAcquisitionRate*1.e6)*(2.0*fRF_frequency)));
            double voltage = amplitude; // square wave is high
             if ( nRFHalfCycles%2==1 ) voltage = -amplitude; // square wave is low
            aSignal->SignalTimeComplex()[index][0] +=  voltage; // in-phase (I)
//            aSignal->SignalTimeComplex()[index][1] += [...]; // quadrature (Q) = 0 for real signals.
        }
*/
```

Uncomment here

18
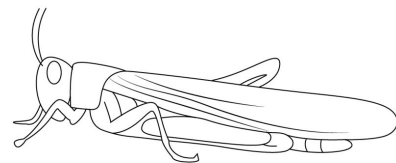
Inside container, compile, run, process:
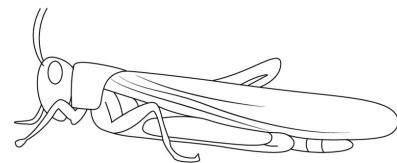
cd /locust_mc/cbuild
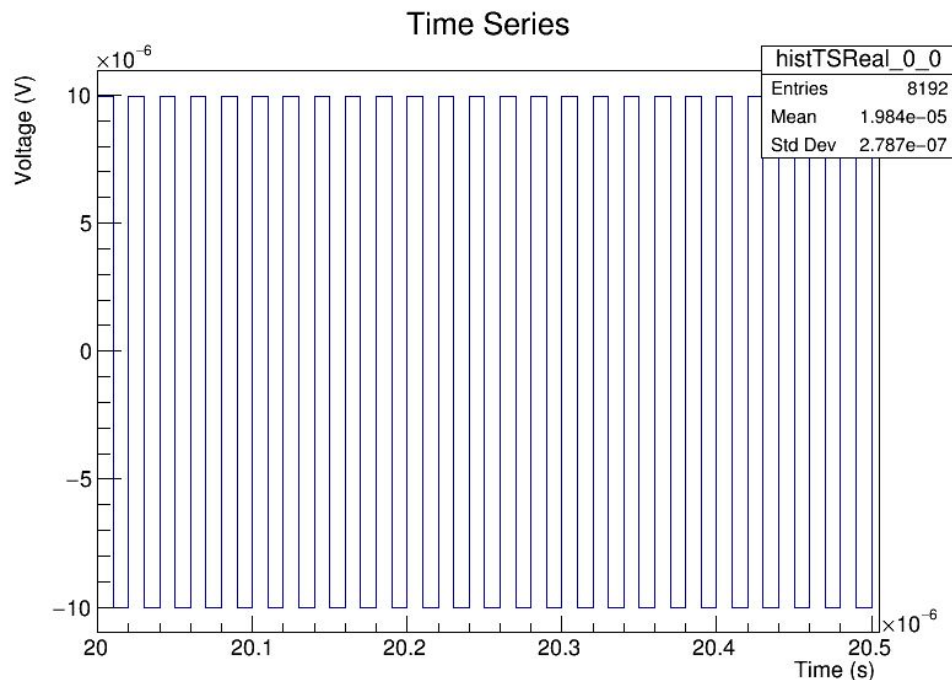
make install

bin/LocustSim config=/work/LocustSquareWaveTemplate.json
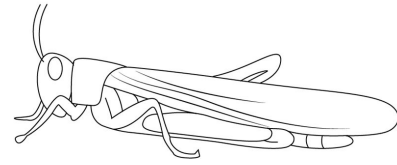
Katydid -c config/katydid.json

Plot Katydid output time series with Root

- In the jupyter browser tab, navigate with single clicks to /work/scripts/plotting/PlotTimeSeries2.ipynb

- Click the ▶ ▶, then click "Restart and run all cells".
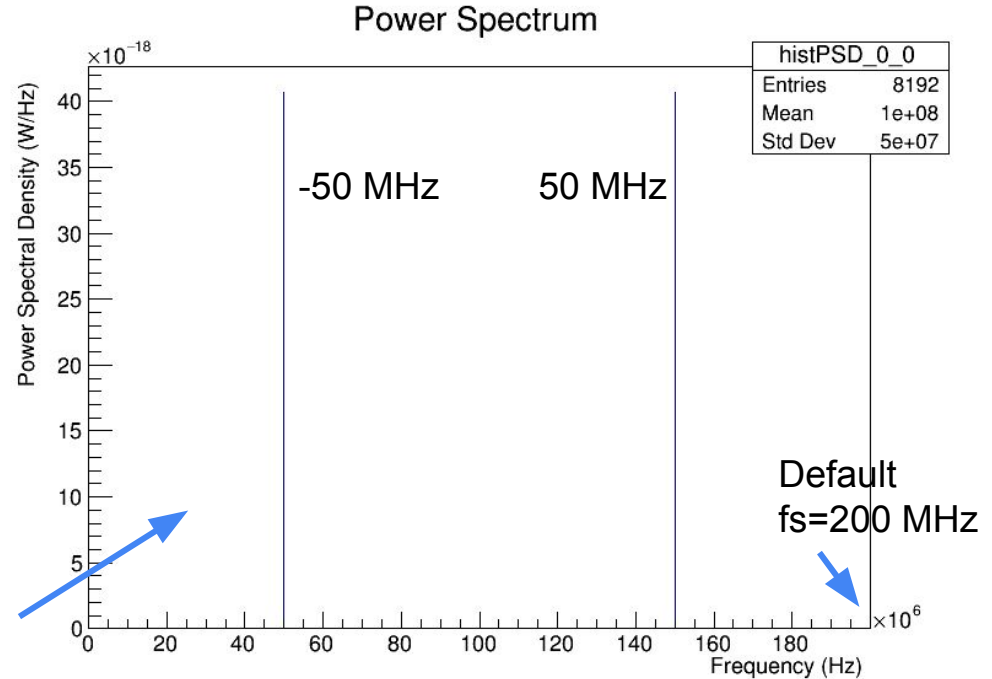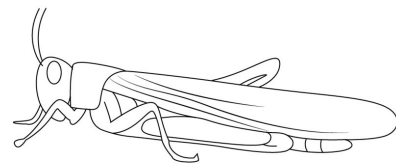
- This plot should appear in the lowest panel   -> .

Plot Katydid frequency spectrum with Root

- In the jupyter browser tab, navigate with single clicks to /work/scripts/plotting/PlotPSD2.ipynb

- Click the ▶▶, then click "Restart and run all cells".

- This plot should appear -> .

This is a symmetric fft spectrum because it comes from a real (not complex) signal. Center corresponds to DC.

Power Spectrum

histPSD_0_0
Entries 8192
Mean 1e+08
Std Dev 5e+07

-50 MHz   50 MHz

Default
fs=200 MHz

Let's parametrize the frequency of the square wave

- This is a convenient way to modify the generator without recompiling the code.
- In LMCSquareWaveSignalGenerator.hh uncomment to add a new private variable:

```
// double fRF_frequency;
```

- In LMCSquareWaveSignalGenerator.cc uncomment to add a new class variable:

```
// fRF_frequency( 10.0e6 )
```

- In LMCSquareWaveSignalGenerator.cc, uncomment this into function Configure():

```
/*
if( aParam.has( "rf-frequency" ) )
{
    fRF_frequency = aParam.get_value< double >( "rf-frequency", fRF_frequency );
}
*/
```
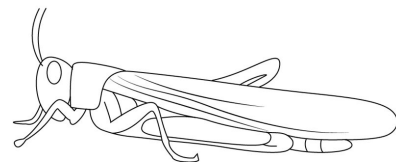
Let's parametrize the frequency of the square wave (cont):

- Replace local hard-wired variable RF_freq with the new parameter fRF_frequency:

  Comment this line: //   double RF_freq = 50.e6; // Hz
  Comment this line: //   int nRFHalfCycles = int(index*(1./(fAcquisitionRate*1.e6)*(2.0*RF_freq)));
  Uncomment this: // int nRFHalfCycles = int(index*(1./(fAcquisitionRate*1.e6)*(2.0*fRF_frequency)));
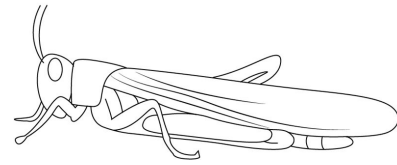
- Compile, run, process:

  cd /locust_mc/cbuild
  make install
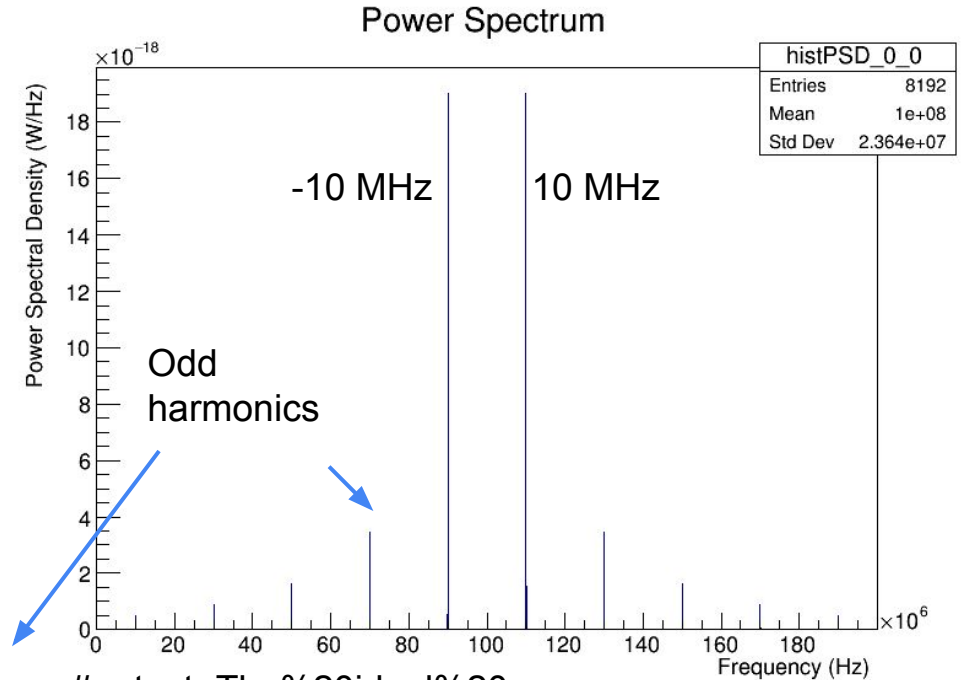 bin/LocustSim config=/work/LocustSquareWaveTemplate.json
  Katydid -c config/katydid.json
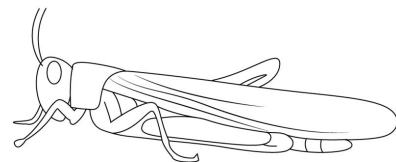
Plot Katydid frequency spectrum with Root

- In the jupyter browser tab, navigate with single clicks to /work/scripts/plotting/PlotPSD2.ipynb

- Click the ▶ ▶, then click "Restart and run all cells".

- This plot should appear -> .

https://en.wikipedia.org/wiki/Square_wave#:~:text=The%20ideal%20square%20wave%20contains,wave%20is%20the%20Gibbs%20phenomenon.

Power Spectrum

-10 MHz   10 MHz

Odd harmonics

Tune the square wave frequency:

- In ~/p8tutorial/locust-tutorial/, edit LocustSquareWaveTemplate.json to add this field:

```
"square-wave":
{
    "rf-frequency": 25.0e6,
},
```
(This will change fRF_frequency from default 10 MHz to 25 MHz.)

- Run, process (without needing to compile):

```
bin/LocustSim config=/work/LocustSquareWaveTemplate.json
Katydid -c config/katydid.json
```

Plot Katydid frequency spectrum with Root

- In the jupyter browser tab, navigate with single clicks to /work/scripts/plotting/PlotPSD2.ipynb

- Click the ▶️▶️, then click "Restart and run all cells".

- This plot should appear -> .

https://en.wikipedia.org/wiki/Square_wave#:~:text=The%20ideal%20square%20wave%20contains,wave%20is%20the%20Gibbs%20phenomenon.

26
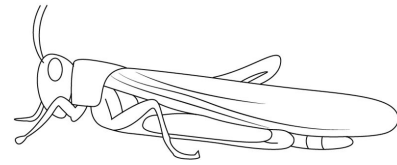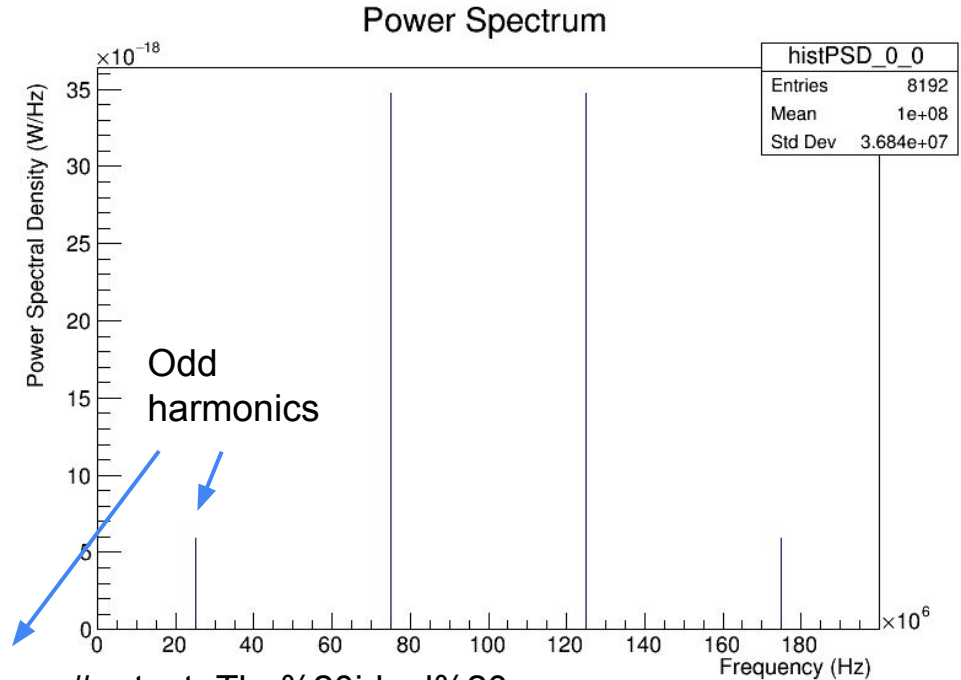
Conclusion and next steps

- In this tutorial we wrote a new subclass of LMCGenerator.

- For additional practice, new configurable subclasses could be added to other parent classes, e.g.:
  - LMCTransmitter.
  - LMCReceiver.
  - LMCPowerCombiner.

- Other unique physics models could also be implemented as a subclass of LMCGenerator.