SC462 Elements of Synthetic Biology: Life 2.0
Prof. Manish K Gupta
DA-IICT

Cello: genetic circuit design automation

Akshay Vijayvergia

Abstract

DNA is the single most important part of all the organisms present on the planet. It consists of the genetic instructions used in the functioning, growth, and development of all known living organisms. Due to this, to study biological architecture, we need to perform computations in living cells by DNA-encoded circuits which indeed process sensory information and control all the biological functions. But due to the various complexions like time and design, these computations become close to impossible. Thus to tackle this problem, we study Cello[3], which serves as a design environment where the user gives a Verilog code as input that is automatically transformed into a DNA sequence providing a genetic circuit design.

Introduction

You could say "I want a cell that can detect certain elements in water and provide different color output signals. Also, the output signals generated should correlate to the concentration of the elements specified detected." .These are the types of problems which could be efficiently simulated by Cello. Cello is a web based open source tool that is used to build genetic circuit diagrams. It's a joint initiative between Boston University and MIT, initiated by the CIDAR lab at Boston.[4]

Its input consists of a Verilog script. This script is then parsed into a truth table which is further synthesized to generate a genetic circuit diagram with the genetically available gate types in the library. For simulating the circuit, a predicted score guides a breadth-first search or a Monte Carlo simulated annealing search. Among all the predicted scores, the one with the highest score is selected. The one which is selected could then be implemented using the various genetic layouts. For finalizing among one or more DNA sequences for the designated circuit, the Eugene language is used which further processes upon the rule based combinatorial design.

To briefly analyze Cello, firstly, we start with the actual/internal working of cello and begin to visualize how the Verilog input is converted to a desired genomic circuit. Later we look into an example and find out the various functionalities provided by the web app.

Working

VERILOG SPPECIFICATION

When opening the website http://www.cellocad.org/verilog.html, we are greeted with a text editor which enables us to input the Verilog script. Cello accepts three forms of Verilog scripts namely case statements, assign statements and structural statements. The Verilog programs usually start with module keyword, followed by the name taking a list of output and input wire names as its arguments. We now focus on all the three specified Verilog types.

1. Case: This style is usually preferred when we directly want to specify the truth table for our circuit.

in1	in2	out1
0	0	0
0	1	1
1	0	1
1	1	0

```
module XOR(output out1, input in1, in2);
  always@(in1,in2)
    begin
      case({in1,in2})
        2'b00: {out1} = 1'b0;
        2'b01: {out1} = 1'b1;
        2'b10: {out1} = 1'b1;
        2'b11: {out1} = 1'b0;
      endcase
    end
endmodule
```

2. Assign: This style is usually preferred when we want to specify circuit using boolean operators. For detailing the order of operations, parentheses are used. One thing should be noted that all the internal wires must be defined before use.

```
module XOR(output out1,  input in1, in2);
  wire w1, w2;
  assign w1 = ~in1 & in2;
  assign w2 = in1 & ~in2;
  assign out1 = w1 | w2;
endmodule
```

3. Structural: This style is usually preferred when we want a gate-level wiring diagram to be specified. Again all the internal wires need to be defined before use.

   Here, within, the parenthesis, the first argument is the output wire and all the other trailing arguments are the inputs. The symbol at the beginning of each line is an operator.

```
module XOR(output out1,  input in1, in2);
  wire w1, w2, w3, w4;
  not (w1, in1);
  not (w2, in2);
  not (w3, in1, w2);
  not (w4, in2, w1);
  or (out1, w3, w4);
endmodule
```

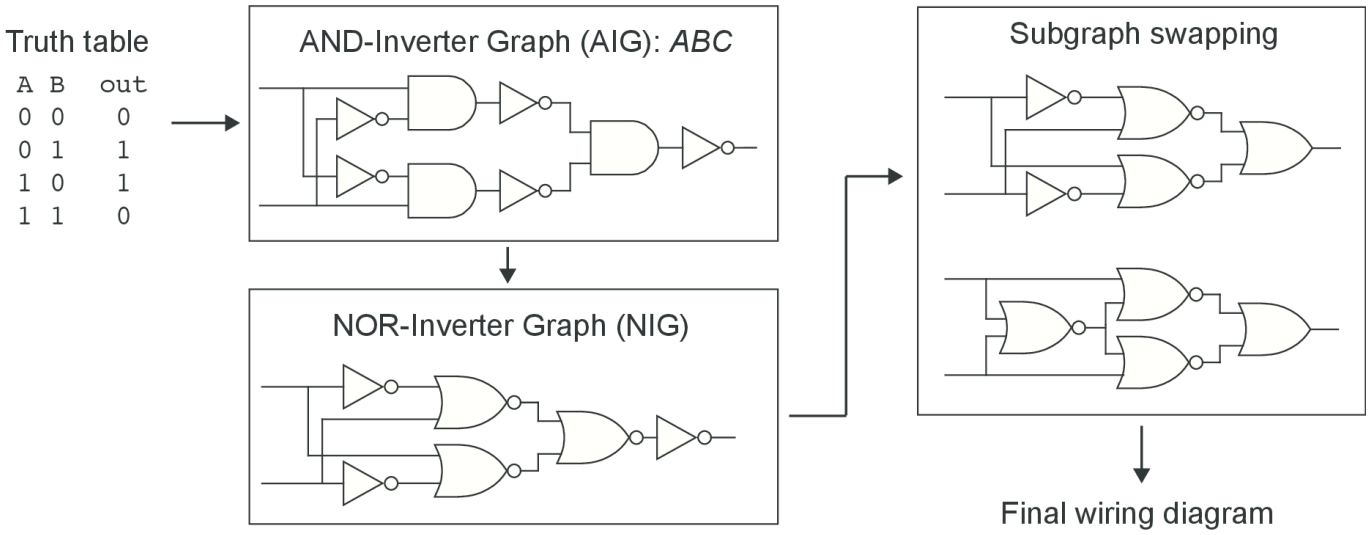
By providing the Verilog input in any of the above three designs, we then move onto the logical synthesis step.

LOGIC SYNTHESIS

The Verilog script is then parsed to generate a truth table. The truth table is then converted to the final wiring diagram in the following steps.

1. AND-Inverter Graph: The table is converted to an AND-Inverted graph consisting of 2-input AND gates and NOT gates.
2. NOR-Inverter Graph: The above is then converted to a NOR-Inverter Graph using DeMorgan's rule
3. Subcircuit substitutions: The above is then simplified and further substituted with smaller but functionally equivalent logic diagrams in order to reduce the number of gates. In this step, other gate types can also be included.[2]

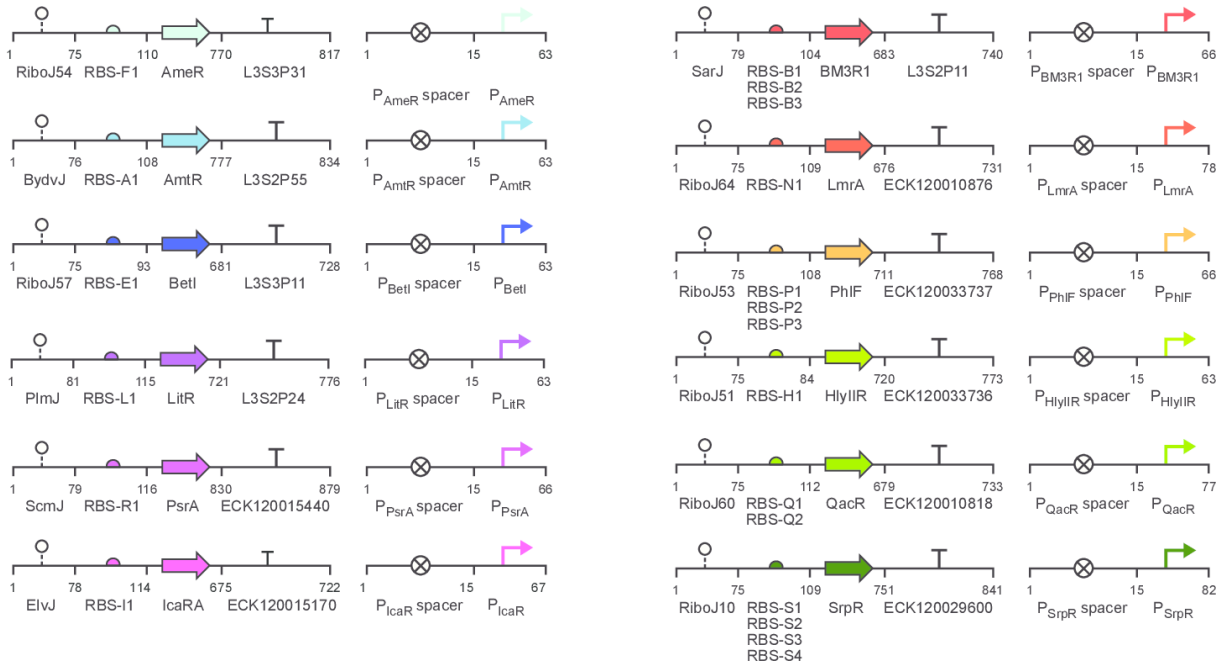
The above process is better described in the figure shown below:



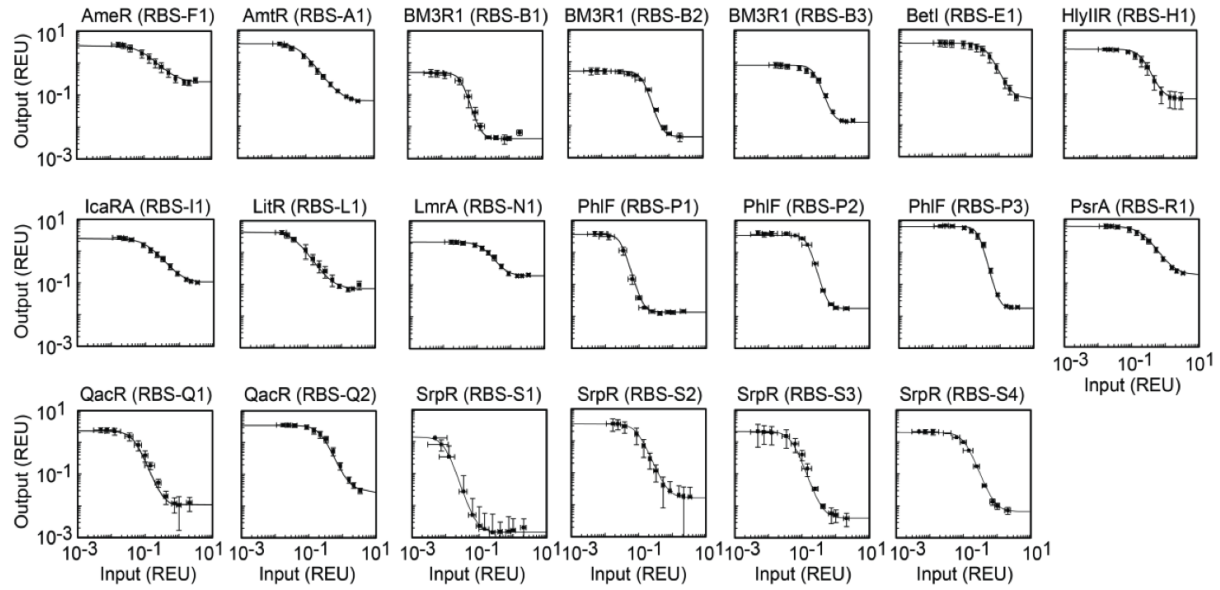
After creating the simplified wiring diagram, we then assign gates to the diagram from the Cello library as per specification detailed in the next subsection.

GATE ASSIGNMENT

Some basic gates in the gate library are shown below. If we want to add other gates to our library, we can add those using the UCF



All the gates shown above have an experimentally determined response function that relates one or more input values to an output value which is specified in standardized units (RPU = Relative Promoter Units). These response functions are then fitted to a Hill function which takes ymax, ymin, K,n as its 4 arguments. The response functions of the above figured gates are shown below:



But still, the main question remains as to how do we assign gates to all the elements in the circuit. This assignment is done using the following assignment algorithms.

1. Breadth-first search: This algorithm guarantees to select a genetic circuit with a global max score among all other assignments. It starts with the gates which are closest to the inputs. The algorithm then goes one gate at a time where signal mismatches are faulty. For each acceptance in the traversal then next breadth first order gate is exhaustively assigned following the same procedure again until all the gates have been visited. After the completion, the circuit with the highest assignment score is selected. [2]

2. Hill climbing: This algorithm starts with a random assignment, and then swapping the assignment of 2 gates. Over here, gate 1 is randomly selected from the circuit to process and gate 2 is selected from the circuit or the gates which are left unused in the gate library for cello. The above specified swap is accepted if the initially initialized circuit score increases. After running this extensively on the whole circuit, the circuit with the highest assignment score is selected.[2]

3. Simulated annealing: This is same as hill climbing except from the difference that the swaps that decrease circuit score with a probability that recombines/anneals over time are selected.[2]

Now for a given assignment, we can have a lot of combination of the arranging the genetic parts into physical diagrams. This problem is solved and assessed using the Eugene specification. The final DNA sequences that result from the Eugene variants is then placed into the genomic locations specified in the earlier describes UCF.

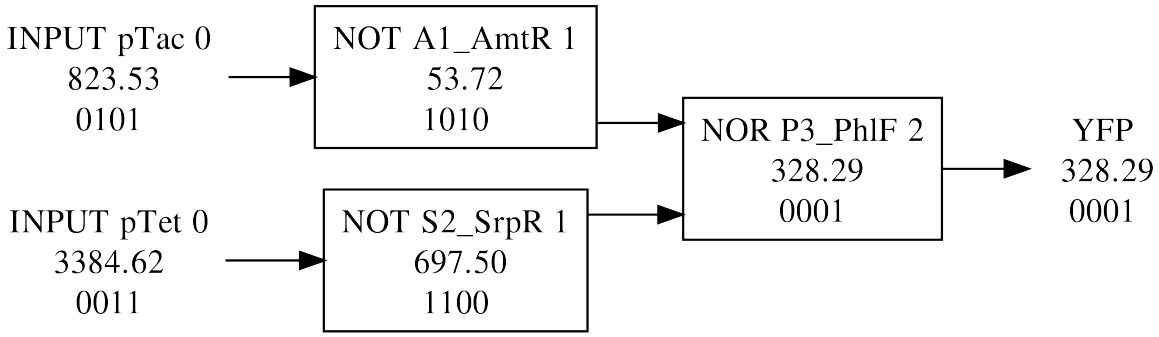
Example

Now let us take a verilog script and run it on cello webapp to find out the various logs and outputs provided by the software.[1]

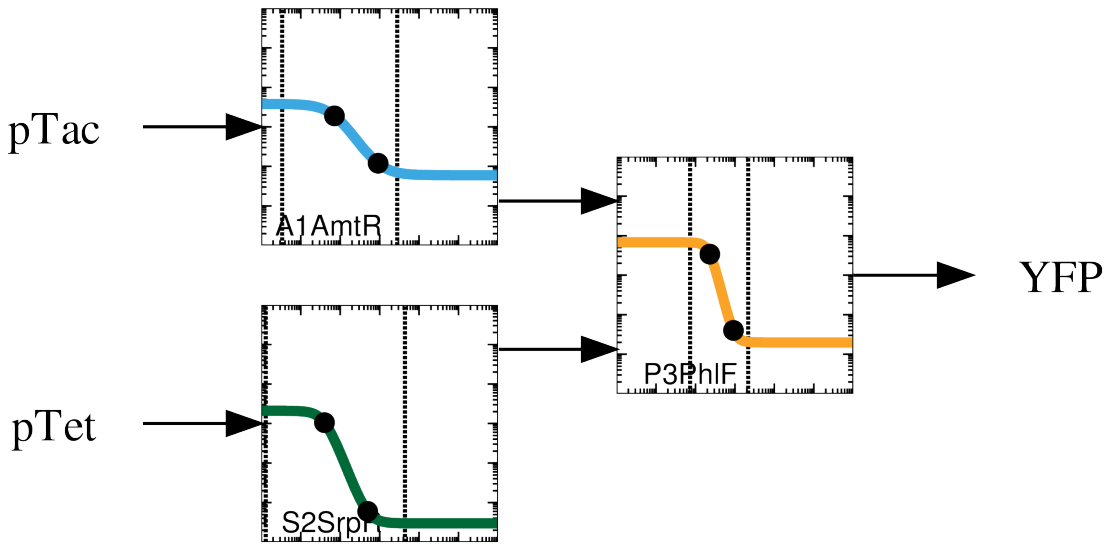
input Verilog

```
module A(output out1,  input in1, in2);
 always@(in1,in2)
  begin
   case({in1,in2})
    2'b00: {out1} = 1'b0;
    2'b01: {out1} = 1'b0;
    2'b10: {out1} = 1'b0;
    2'b11: {out1} = 1'b1;
   endcase
  end
endmodule
```

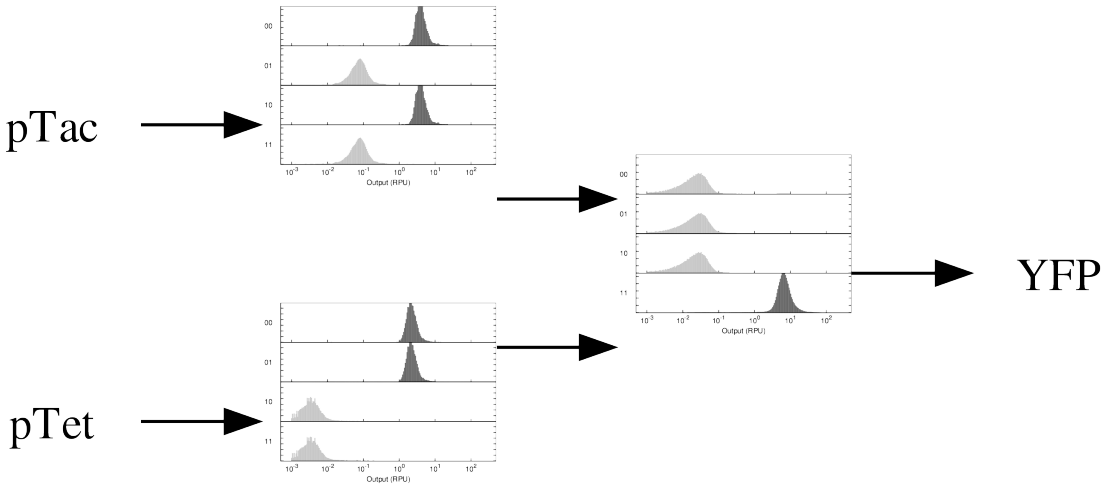
On running the above case script for AND gate and selecting the inputs as pTac and pTet along with the output as RPF, we firstly get the following gate assignment after its logical synthesis as shown below:



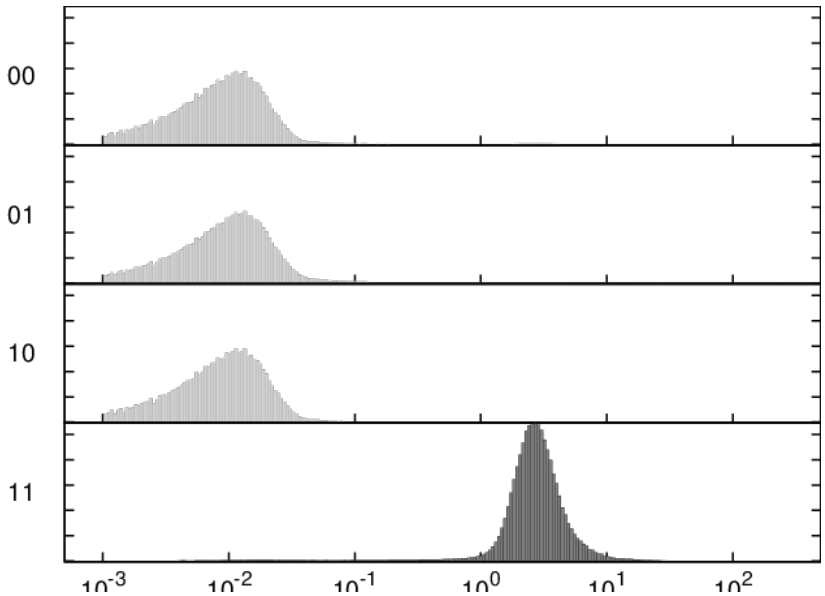
The Response function of the above gate assignment circuit is shown below:



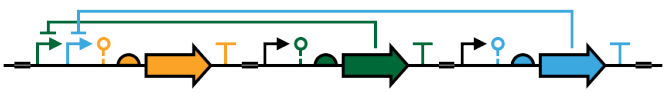
As stated before, in the gate assignment subsection, all the gates have a predefined output in RPUs(Relative Promoter Units). For the circuit, we see the predicted RPUs of the gates as shown below.



On providing the assignment algorithms we get the predicted gate output as shown below:



The final circuit diagram created by the software is shown below after processing through the Eugene rules:



In all the steps shown above, log files are also generated which further provide a detailed understanding of all the above steps as simulated by the Cello software. In the end we are greeted with sbol and plasmid log files which are indeed the results that we needed. Some snapshots of the logs files are attached below for reference

```
LOCUS       test1_AND_A000_plasmid_circuit_P000  7741 bp ds-DNA   circular     05-Mar-2017
FEATURES             Location/Qualifiers
     backbone        1..53
                     /label=backbone
                     /ApEinfo_fwdcolor=pink
                     /ApEinfo_revcolor=pink
     scar            54..57
                     /label=Escar
                     /ApEinfo_fwdcolor=gray
                     /ApEinfo_revcolor=gray
     promoter        58..139
                     /label=pSrpR
                     /ApEinfo_fwdcolor=green
                     /ApEinfo_revcolor=green
     promoter        140..202
                     /label=pAmtR
                     /ApEinfo_fwdcolor=green
                     /ApEinfo_revcolor=green
     ribozyme        203..281
                     /label=RiboJ53
                     /ApEinfo_fwdcolor=magenta
                     /ApEinfo_revcolor=magenta
     rbs             282..299
                     /label=P3
                     /ApEinfo_fwdcolor=blue
                     /ApEinfo_revcolor=blue
     cds             300..902
                     /label=PhlF
                     /ApEinfo_fwdcolor=cyan
                     /ApEinfo_revcolor=cyan
     terminator      903..959
                     /label=ECK120033737
                     /ApEinfo_fwdcolor=red
                     /ApEinfo_revcolor=red
     scar            960..963
                     /label=Xscar
                     /ApEinfo_fwdcolor=gray
                     /ApEinfo_revcolor=gray
     promoter        964..1037
```

```
     rbs             2042..2075
                     /label=A1
                     /ApEinfo_fwdcolor=blue
                     /ApEinfo_revcolor=blue
     cds             2076..2744
                     /label=AmtR
                     /ApEinfo_fwdcolor=cyan
                     /ApEinfo_revcolor=cyan
     terminator      2745..2801
                     /label=L3S2P55
                     /ApEinfo_fwdcolor=red
                     /ApEinfo_revcolor=red
     scar            2802..2805
                     /label=Cscar
                     /ApEinfo_fwdcolor=gray
                     /ApEinfo_revcolor=gray
     backbone        2806..5882
                     /label=sensor_module
                     /ApEinfo_fwdcolor=pink
                     /ApEinfo_revcolor=pink
     backbone        5883..7741
                     /label=backbone
                     /ApEinfo_fwdcolor=pink
                     /ApEinfo_revcolor=pink
ORIGIN
        1 CCAATTATTG AAGGCCTCCC TAACGGGGGG CCTTTTTTTG TTTCTGGTCT CCCgcttTCT
       61 ATGATTGGTC CAGATTCGTT ACCAATTGAC AGCTAGCTCA GTCCTAGGTA TATACATACA
      121 TGCTTGTTTG TTTGTAAACC TTGTCCAACC AAATGATTCG TTACCAATTG ACAGTTTCTA
      181 TCGATCTATA GATAATGCTA GCCTGAAGCG GTCAACGCAT GTGCTTTGCG TTCTGATGAG
      241 ACAGTGATGT CGAAACCGCC TCTACAAATA ATTTTGTTTA ACTTTACGAG GGCGATCCTA
      301 TGGCACGTAC CCCGAGCCGT AGCAGCATTG GTAGCCTGCG TAGTCCGCAT ACCCATAAAG
      361 CAATTCTGAC CAGCACCATT GAAATCCTGA AAGAATGTGG TTATAGCGGT CTGAGCATTG
      421 AAAGCGTTGC ACGTCGTGCC GGTGCAAGCA AACCGACCAT TTATCGTTGG TGGACCAATA
      481 AAGCAGCACT GATTGCCGAA GTGTATGAAA ATGAAAGCGA ACAGGTGCGT AAATTTCCGG
      541 ATCTGGGTAG CTTTAAAGCC GATCTGGATT TTCTGCTGCG TAATCTGTGG AAAGTTTGGC
      601 GTGAAACCAT TTGTGGTGAA GCATTTCGTT GTGTTATTGC AGAAGCACAG CTGGACCCTG
      661 CAACCCTGAC CCAGCTGAAA GATCAGTTTA TGGAACGTCG TCGTGAGATG CCGAAAAAAC
      721 TGGTTGAAAA TGCCATTAGC AATGGTGAAC TGCCGAAAGA TACCAATCGT GAACTGCTGC
      781 TGGATATGAT TTTTGGTTTT TGTTGGTATC GCCTGCTGAC CGAACAGCTG ACCGTTGAAC
      841 AGGATATTGA AGAATTTACC TTCCTGCTGA TTAATGGTGT TTGTCCGGGT ACACAGCGTT
```

References

[1] Cellocad. Cellocad, 2017.

[2] Cidarlab/Cello. Cello, 2016.

[3] ncbiRefs. Genetic circuit design automation. - pubmed - ncbi, 2017.

[4] Reddit/cerenity. Mit develops cello, 2016.