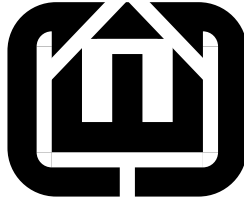


CEC GRC simulator



Stephen A. Edwards, Jia Zeng, Cristian Soviani
Columbia University
sedwards@cs.columbia.edu

Abstract

This simulates much of the GRC format. It was designed primarily for testing and is not terribly efficient.

Contents

1	Visitor Methods	2
1.1	Fork	2
1.2	Terminate	2
1.3	Sync	2
1.4	Enter	3
1.5	Suspend	5
1.6	Actions: Emit, Exit, Assign, Startcounter, Nop	5
1.7	Switch	7
1.8	Test	8
1.9	Expressions	8
1.10	DefineSignal	10
2	The GRCsim Class	11
3	Helper Methods	13
3.1	schedule and schedule_child	13
3.2	intVal	13
3.3	doswitch	13
3.4	dfs	14
3.5	execute max	15
3.6	Execute Vectors	16

3.7	init	18
3.8	Clear Inputs	19
3.9	dotick	19
4	Top-level files	20

1 Visitor Methods

1.1 Fork

Fork schedules all its children.

2a $\langle \text{method declarations } 2a \rangle \equiv$ (11) 2c \triangleright
`Status visit(Fork &);`

2b $\langle \text{method definitions } 2b \rangle \equiv$ (20b) 2d \triangleright
`Status GRCsim::visit(Fork &s)`
`{`
`if (debug) cerr << cfgmap[&s] << ":fork" << '\n';`
`for (vector<GRCNode *>::iterator i = s.successors.begin() ;`
`i != s.successors.end() ; i++) schedule(*i);`
`return Status();`
`}`

1.2 Terminate

A Terminate node simply schedules its children.

2c $\langle \text{method declarations } 2a \rangle + \equiv$ (11) <2a 2e \triangleright
`Status visit(Terminate &);`

2d $\langle \text{method definitions } 2b \rangle + \equiv$ (20b) <2b 3a \triangleright
`Status GRCsim::visit(Terminate &s)`
`{`
`if (debug) cerr << cfgmap[&s] << ":term " << s.code << endl;`
`schedule_child(&s);`
`return Status();`
`}`

1.3 Sync

The Sync node computes the maximum exit level among all its predecessors (these should be Terminate nodes by construction) and schedules its corresponding child.

2e $\langle \text{method declarations } 2a \rangle + \equiv$ (11) <2c 3b \triangleright
`Status visit(Sync &);`

3a $\langle \text{method definitions } 2b \rangle + \equiv$ (20b) $\langle 2d \ 4 \rangle$

```

Status GRCSim::visit(Sync &s)
{
    int max_level = -1;
    assert(s.dataPredecessors.size() > 0); // grcdps should have computed them
    for ( vector<GRNode *>::const_iterator i = s.dataPredecessors.begin() ;
          i != s.dataPredecessors.end() ; i++ ) {
        Terminate *t = dynamic_cast<Terminate*>(*i);
        assert(t);
        if (debug & debugSync)
            if (debug) cerr << cfgmap[&s] << ":sync checking "
                          << cfgmap[t] << ":term " << t->code << '\n';
        if ((sched[t]) && (t->code > max_level)) max_level = t->code;
    }

    if (debug) cerr << cfgmap[&s] << ":sync at level " << max_level << '\n';

    assert(max_level >= 0);
    assert(max_level < (int) s.successors.size());
    schedule(s.successors[max_level]);

    return Status();
}

```

1.4 Enter

The key operation for the Enter node is walking up the selection tree to find the closest exclusive node whose child this node is.

3b $\langle \text{method declarations } 2a \rangle + \equiv$ (11) $\langle 2e \ 5a \rangle$

```

void setState(STNode *n);

```

```

4  <method definitions 2b>+≡
    void GRCsim::setState(STNode *n)
    {
        STexcl *exclusive = 0;

        for (;;) {
            assert(n);

            STNode *parent = n->parent;

            if (dynamic_cast<STpar*>(parent) != NULL) {
                if (debug) cerr << "parent of node " << stmap[n]
                    << " is a parallel node" << endl;
                return;
            }

            exclusive = dynamic_cast<STexcl*>(parent);
            if (exclusive != NULL) break; // found the exclusive node
            n = parent;
        }

        assert(exclusive != NULL);

        // Locate node n among the children of "parent"

        vector<STNode*>::iterator i = exclusive->children.begin();
        while (*i != n && i != exclusive->children.end()) i++;

        assert(i != exclusive->children.end());

        int childnum = i - exclusive->children.begin();

        assert(childnum >= 0);

        // Check for program termination

        if (exclusive == stroot) {
            STleaf *lf = dynamic_cast<STleaf *>(n);
            if ((lf) && (lf->isfinal())) {
                ISFinal = true;
                if (debug) cerr << "program terminated\n";
            }
        }

        state[exclusive] = childnum;
        if (debug) cerr << "    state[" << stmap[exclusive] << "] = "
            << childnum << endl;
    }

```

The visitor for the Enter node marks itself as selected using `setState` and schedules its children.

5a *<method declarations 2a>+≡* (11) <3b 5c>
`Status visit(Enter &);`

5b *<method definitions 2b>+≡* (20b) <4 5d>
`Status GRCsim::visit(Enter &s)`
`{`
 `if (debug) cerr << cfgmap[&s] << ":enter " << stmap[s.st];`
 `setState(s.st);`
 `schedule_child(&s);`
 `return Status();`
`}`

1.5 Suspend

5c *<method declarations 2a>+≡* (11) <5a 5e>
`Status visit(STSuspend &);`

This just schedules its child.

5d *<method definitions 2b>+≡* (20b) <5b 5f>
`Status GRCsim::visit(STSuspend &s)`
`{`
 `if (debug) cerr << cfgmap[&s] << ":suspend " << stmap[s.st];`
 `schedule_child(&s);`
 `return Status();`
`}`

1.6 Actions: Emit, Exit, Assign, Startcounter, Nop

5e *<method declarations 2a>+≡* (11) <5c 10b>
`Status visit(Action &);`
`Status visit(Emit &);`
`Status visit(Exit &);`
`Status visit(Assign &);`
`Status visit(StartCounter &);`
`Status visit(Nop &);`

Action GRC nodes simply invoke their bodies.

5f *<method definitions 2b>+≡* (20b) <5d 6a>
`Status GRCsim::visit(Action &s)`
`{`
 `if (debug) cerr << cfgmap[&s];`
 `s.body->welcome(*this);`
 `schedule_child(&s);`
 `return Status();`
`}`

Emit actions simply set their signal's status to present.

```
6a  <method definitions 2b>+≡ (20b) <5f 6b>
    Status GRCsim::visit(Emit &e)
    {
        assert(e.signal);
        SignalSymbol *ss = e.signal;
        if (debug) cerr << ":emit " << ss->name << endl;
        signals[ss] = true;

        return Status();
    }
```

Exit actions are similar to Emit.

```
6b  <method definitions 2b>+≡ (20b) <6a 6c>
    Status GRCsim::visit(Exit &e)
    {
        assert(e.trap);
        SignalSymbol *ss = e.trap;
        if (debug) cerr << ":exit " << ss->name << endl;
        signals[ss] = true;

        return Status();
    }
```

Assign statement set their variable to the value of their expression.

```
6c  <method definitions 2b>+≡ (20b) <6b 7a>
    Status GRCsim::visit(Assign &a)
    {
        assert(a.variable);
        assert(a.value);
        assert(a.variable->type);

        if (a.variable->type->name != "integer" &&
            a.variable->type->name != "boolean") {
            throw IR::Error("Only integer and boolean variables supported");
        }

        var[a.variable] = intVal(a.value);

        return Status();
    }
```

StartCounter actions reset their counter.

7a *<method definitions 2b>+≡* (20b) <6c 7b>

```

Status GRCSim::visit(StartCounter &stcnt)
{
    if (debug) cerr << ":start counter\n";
    assert(stcnt.counter);
    assert(stcnt.count);
    counters[stcnt.counter] = intVal(stcnt.count);
    return Status();
}

```

NOP nodes do, not surprisingly, nothing.

7b *<method definitions 2b>+≡* (20b) <7a 7c>

```

Status GRCSim::visit(Nop &n)
{
    if (debug) cerr << ":nop\n";
    schedule_child(&n);
    return Status();
}

```

1.7 Switch

7c *<method definitions 2b>+≡* (20b) <7b 8a>

```

Status GRCSim::visit(Switch &s)
{
    assert(s.st);
    int act_ch = doswitch(s.st);
    assert(act_ch < (int) s.successors.size());
    GRNode *child = s.successors[act_ch];
    if (debug) cerr << cfgmap[&s] << ":switch " << stmap[s.st]
        << " --" << act_ch << "--> "
        << (child ? cfgmap[child] : -1) << '\n';
    if (child) schedule(child);

    return Status();
}

```

1.8 Test

8a (20b) <7c 8b>

```

<method definitions 2b>+≡
Status GRCsim::visit(Test &s)
{
    assert(s.predicate);
    assert(s.successors.size()==2);
    int predvalue = intVal(s.predicate);

    GRCNode *successor = s.successors[predvalue ? 1 : 0];

    if (debug) {
        cerr << cfgmap[&s] << ":test --" << predvalue << "--> ";
        if (successor) cerr << cfgmap[successor];
        else cerr << "(none)";
        cerr << '\n';
    }

    if (successor) schedule(successor);

    return Status();
}

```

1.9 Expressions

8b (20b) <8a 8c>

```

<method definitions 2b>+≡
Status GRCsim::visit(Literal &s)
{
    int val;
    if ( sscanf(s.value.c_str(), "%d", &val) != 1 ) assert(0);
    return Status(val);
}

```

8c (20b) <8b 8d>

```

<method definitions 2b>+≡
Status GRCsim::visit(LoadSignalExpression &lse)
{
    SignalSymbol *ss = lse.signal;
    assert(ss);
    return Status(signals[ss]);
}

```

8d (20b) <8c 9a>

```

<method definitions 2b>+≡
Status GRCsim::visit(LoadVariableExpression &lve)
{
    VariableSymbol *vs = lve.variable;
    assert(vs);
    if (var.find(vs) == var.end())
        throw IR::Error("reading uninitialized variable " + vs->name);
    return Status(var[vs]);
}

```


9a $\langle \text{method definitions 2b} \rangle + \equiv$ (20b) $\langle 8d \ 9b \rangle$

```

Status GRCSim::visit(CheckCounter &chkcnt)
{
    assert(chkcnt.counter);
    assert(chkcnt.predicate);
    if (intVal(chkcnt.predicate))
        counters[chkcnt.counter]--;

    return Status( counters[chkcnt.counter] == 0 );
}

```

9b $\langle \text{method definitions 2b} \rangle + \equiv$ (20b) $\langle 9a \ 9c \rangle$

```

Status GRCSim::visit(BinaryOp &e)
{
    assert(e.source1);
    assert(e.source2);
    int val1 = intVal(e.source1);
    int val2 = intVal(e.source2);

    int result =
        (e.op == "and") ? (val1 && val2) :
        (e.op == "or") ? (val1 || val2) :
        (e.op == "+") ? (val1 + val2) :
        (e.op == "-") ? (val1 - val2) :
        (e.op == "*") ? (val1 * val2) :
        (e.op == "/") ? (val1 / val2) :
        (e.op == "mod") ? (val1 % val2) :
        (e.op == "=") ? (val1 == val2) :
        (e.op == "<") ? (val1 != val2) :
        (e.op == "<") ? (val1 < val2) :
        (e.op == "<=") ? (val1 <= val2) :
        (e.op == ">") ? (val1 > val2) :
        (e.op == ">=") ? (val1 >= val2) :
        0;

    return Status(result);
}

```

9c $\langle \text{method definitions 2b} \rangle + \equiv$ (20b) $\langle 9b \ 10a \rangle$

```

Status GRCSim::visit(UnaryOp &e)
{
    int val = intVal(e.source);

    int result =
        (e.op == "not") ? ( !val ) :
        (e.op == "-") ? ( -val ) :
        0;

    return Status(result);
}

```

```

10a  <method definitions 2b>+≡ (20b) <9c 10c>
      Status GRCsim::visit(FunctionCall &)
      {
        throw IR::Error("Function calls are not supported");
      }

```

1.10 DefineSignal

A DefineSignal node resets its signal's presence. This is used to initialize local signals when they enter scope.

```

10b  <method declarations 2a>+≡ (11) <5e 13b>
      Status visit(DefineSignal &);

10c  <method definitions 2b>+≡ (20b) <10a 13a>
      Status GRCsim::visit(DefineSignal &d)
      {
        assert(d.signal);
        SignalSymbol *ss = d.signal;
        assert(ss);
        if (debug) cerr << cfgmap[&d] << ":DefineSignal " << ss->name << '\n';
        signals[ss] = false;
        schedule_child(&d);
        return Status();
      }

```

2 The GRCsim Class

This is the heart of the simulator. Derived from the `Visitor` class, its methods simulate each type of GRC node.

```

11   $\langle \text{GRCsim class 11} \rangle \equiv$  (20a)
    class GRCsim: public Visitor {

        int debug;
        static const int debugDFS = 1 << 1;
        static const int debugSync = 1 << 2;
        static const int debugVectors = 1 << 3;
        int useold;

        GRCgraph *top;
        EnterGRC *entergrc;
        SymbolTable *sigs;
        STNode *stroot;
        STleaf *boot;
        GRCNode *grcroot;

        Module *module;

        GRCNode::NumMap &cfgmap;
        STNode::NumMap &stmap;
        map<STNode*, int> state; // State information for selection tree nodes
        vector<GRCNode*> topo;
        map<GRCNode*, bool> sched;
        set<GRCNode*> dfs_notwhite, dfs_black;
        map<SignalSymbol*, bool> signals; // Status of each signal
        map<Counter*, int> counters; // Value of each counter
        map<VariableSymbol*, int> var; // Value of each variable

        std::ostream &outf;
        bool ISFinal;
        bool jump;

    public:
        GRCsim(GRCgraph *top, Module *m, GRCNode::NumMap &cm,
              STNode::NumMap &sm, int db, std::ostream &outf)
            : debug(db), top(top), sigs(m->signals),
              module(m), cfgmap(cm), stmap(sm), outf(outf) {}

        virtual ~GRCsim() {}

        void visit (GRCNode *);

        Status visit(EnterGRC &) { return Status(); }
        Status visit(ExitGRC &) { return Status(); }
        Status visit(Switch &);

```

```
Status visit(Test &);
Status visit(LoadSignalExpression &);
Status visit(CheckCounter &);
Status visit(BinaryOp &);
Status visit(UnaryOp &);
Status visit(LoadSignalValueExpression &) { assert(0); return Status(); }
Status visit(FunctionCall &);
Status visit(LoadVariableExpression &);
Status visit(Literal &);

Status visit(STexcl &) { return Status(); }
Status visit(STref &) { return Status(); }
Status visit(STpar &) { return Status(); }
Status visit(STleaf &) { return Status(); }

void dfs(GRCNode *n);
void init();

void schedule_child(GRCNode *);
void schedule(GRCNode *);
void clear_inputs();
void dotick();
void execute_max(int);
void execute_vectors(std::istream &);
int doswitch(STNode *n);

<method declarations 2a>
};
```

3 Helper Methods

3.1 schedule and schedule_child

Schedule adds the given node to the “to be executed” map. `schedule_child` schedules a node’s child, if one exists.

13a *<method definitions 2b>+≡* (20b) <10c 13c>

```

void GRCsim::schedule(GRCNode *n)
{
    if (n) sched[n] = true;
}

void GRCsim::schedule_child(GRCNode *n)
{
    assert(n);
    assert(n->successors.size() <= 1);
    if ( !n->successors.empty() )
        schedule(n->successors.front());
}

```

3.2 intVal

This evaluates what is expected to be an integer or Boolean expression and returns the result.

13b *<method declarations 2a>+≡* (11) <10b>

```

int intVal(ASTNode *n)
{
    assert(n);
    Status s = n->welcome(*this);
    return s.i; // Integer-valued result
}

```

3.3 doswitch

Returns the number of the currently-active child of a Switch node in the selection tree.

13c *<method definitions 2b>+≡* (20b) <13a 14>

```

int GRCsim::doswitch(STNode *n)
{
    if ( (dynamic_cast<STexcl*>(n)) == NULL )
        cerr << "error: testing non-excl node\n";
    if ( state.find(n) == state.end() )
        cerr << "error: testing uninitialized state for node " << stmap[n] << endl;
    assert(state[n] >= 0);
    assert(state[n] < (int) n->children.size());
    return state[n];
}

```

3.4 dfs

Order the nodes in the control-flow graph.

FIXME: Not all constructive programs can be scheduled statically; some will need data-dependent dynamic scheduling.

```

14  <method definitions 2b>+≡ (20b) <13c 15>
    void GRCsim::dfs(GRCNode *n){
        vector<GRCNode *>::iterator i;

        if (!n) return;

        if ( debug & debugDFS ) cerr << "visiting " << cfgmap[n] << "...";
        if ( dfs_notwhite.count(n) > 0 ) {
            if (debug & debugDFS ) cerr << "visited before\n";
            if (dfs_black.count(n) == 0) {
                cerr << "GRCsim: cycle detected\n";
                exit(100);
            }
            return;
        }
        dfs_notwhite.insert(n);

        if (jump) {
            if (debug & debugDFS ) cerr << "JUMP!\n";
            jump = false;
        }

        if ( debug & debugDFS ) cerr << "visiting\n";

        if ( debug & debugDFS ) cerr << "dfs children\n";
        for (i = n->successors.begin(); i != n->successors.end(); i++)
            dfs(*i);

        if (n->dataSuccessors.size()>0){
            if ( debug & debugDFS ) cerr << "dfs datadps children\n";
            for (i = n->dataSuccessors.begin();
                i != n->dataSuccessors.end(); i++){
                jump = true;
                dfs(*i);
                jump = true;
            }
        }

        assert(n);
        topo.push_back(n);
        dfs_black.insert(n);
        if ( debug & debugDFS ) cerr << "push back " << cfgmap[n] << "\n";
    }

```

3.5 execute max

A simulation dispatch method. Initialize the simulation and run the program for either the given number or ticks or until the program terminates, whichever is sooner.

```

15  <method definitions 2b>+≡ (20b) <14 16>
    void GRCsim::execute_max(int maxticks)
    {
        int ntick;

        useold=0;

        init();
        ntick=0;
        ISFinal = false;

        do {
            if (debug) cerr << "##### TICK " << ntick << endl;
            char buf[5];
            sprintf(buf, "%4d ", ntick);
            outf << buf;
            ntick++;
            clear_inputs();
            dotick();
        } while ((!ISFinal) && (maxticks < 0 || (ntick <= maxticks)));
    }

```

3.6 Execute Vectors

A simulation dispatch method. Initialize the simulation and run the program, taking input signals presence/absence information from a test vector file.

```

16  <method definitions 2b>+≡ (20b) <15 18>
    void GRCsim::execute_vectors(std::istream &vf)
    {
        // Enumerate the inputs to be read from the vector file

        vector<SignalSymbol*> inputs;

        for ( SymbolTable::const_iterator isym = sigs->begin() ;
              isym != sigs->end() ; isym++ ) {
            SignalSymbol *ss = dynamic_cast<SignalSymbol*>(*isym);
            assert(ss);
            if ( ss->name != "tick" && (ss->kind == SignalSymbol::Input ||
                                         ss->kind == SignalSymbol::Inputoutput ) ) {
                if ( debug & debugVectors ) std::cout << ss->name << '\n';
                inputs.push_back(ss);
            }
        }

        init();

        int ntick = 0;
        ISFinal = false;
        do {
            string line;
            do {
                vf >> line;
            } while (line.size() == 0 && !vf.eof());
            if (vf.eof()) break;

            if ( line.size() < inputs.size() ) {
                cerr << "Not enough inputs (" << line.size() << ' ' << inputs.size()
                     << ") in test vector file\n";
                exit(-2);
            }

            clear_inputs();
            string::const_iterator j = line.begin();
            for ( vector<SignalSymbol*>::const_iterator i = inputs.begin() ;
                  i != inputs.end() ; i++, j++ )
                signals[*i] = (*j) == '1';

            if (debug) cerr << "##### TICK " << ntick << endl;
            char buf[5];
            sprintf(buf, "%4d ", ntick);
            outf << buf;
            ntick++;
        }
    }

```



```
        dotick();  
    } while (!ISFinal);  
}
```

3.7 init

Initialize the simulation. Gather special nodes in the graph and run DFS to order them.

```

18  <method definitions 2b>+≡ (20b) <16 19a>
    void GRCsim::init()
    {
        entergrc = dynamic_cast<EnterGRC*>(top->control_flow_graph);
        if (entergrc)
            grcroot = dynamic_cast<GRCNode*>(entergrc->successors.back());
        else
            grcroot = dynamic_cast<GRCNode*>(top->control_flow_graph);
        assert (grcroot);
        stroot = dynamic_cast<STexcl *>(top->selection_tree);
        assert(stroot);
        boot = dynamic_cast<STleaf* >(stroot->children.back());
        assert(boot);

        setState(boot);

        if ( debug & debugDFS ) cerr << "will dfs\n";
        jump = false;
        dfs(grcroot);
        if (debug & debugDFS ) cerr << "init ok\n";

        if (debug & debugDFS ){
            cerr << "-----topo-----:\n";
            for (vector<GRCNode*>::iterator i = topo.end()-1; i >= topo.begin(); i--)
                cerr << cfgmap[*i] << ' ';
            cerr << "\n";
        }

        // Initialize constants

        assert(module->constants);
        for ( SymbolTable::const_iterator i = module->constants->begin();
              i != module->constants->end() ; i++ ) {
            ConstantSymbol *cs = dynamic_cast<ConstantSymbol*>(*i);
            if (cs && cs->initializer) {
                assert(cs->type);
                if (cs->type->name != "integer" &&
                    cs->type->name != "boolean")
                    throw IR::Error("only integer and boolean constants are supported");
                var[cs] = intVal(cs->initializer);
            }
        }
    }
}

```

3.8 Clear Inputs

Reset all the signals; set the tick signal to be present.

```
19a  <method definitions 2b>+≡ (20b) <18 19b>
      void GRCsim::clear_inputs()
      {
        for( SymbolTable::const_iterator isym = sigs->begin() ; isym != sigs->end() ;
            isym++ ) {
          SignalSymbol *ss = dynamic_cast<SignalSymbol*>(*isym);
          if (ss) signals[ss] = (ss->name == "tick");
        }
      }
```

3.9 dotick

Simulate the program for a single cycle.

```
19b  <method definitions 2b>+≡ (20b) <19a>
      void GRCsim::dotick()
      {
        int tsz = topo.size();

        for (int i = 0 ; i < tsz ; i++) sched[topo[i]] = false;

        sched[ topo[tsz-1] ] = true;

        for ( int i = tsz - 1 ; i >= 0 ; i-- ) {
          assert(topo[i]);
          if ( sched[topo[i]] )
            topo[i]->welcome(*this);
        }

        for ( SymbolTable::const_iterator isym = sigs->begin() ;
            isym != sigs->end() ; isym++ ) {
          SignalSymbol *ss = dynamic_cast<SignalSymbol*>(*isym);
          assert(ss);
          if (debug) cerr << ss->name << " : " << signals[ss] << "\n";
          if ( ss->kind == SignalSymbol::Output )
            outf << ss->name << "=" << (signals[ss] ? '1' : '0') << " ";
          else if ( ss->kind == SignalSymbol::Inputoutput )
            outf << ss->name << "_IO_0=" << (signals[ss] ? '1' : '0') << " ";
        }
        outf << "\n";
      }
```

4 Top-level files

```
20a  <GRCsim.hpp 20a>≡
      #ifndef _GRCSIM_HPP
      #  define _GRCSIM_HPP

      #  include "IR.hpp"
      #  include "AST.hpp"
      #  include <iostream>
      #  include <stdlib.h>
      #  include <stdio.h>
      #  include <set>
      #  include <map>

      namespace AST {

      using std::set;
      using std::map;

      <GRCsim class 11>

      }
      #endif

20b  <GRCsim.cpp 20b>≡
      #include "GRCsim.hpp"

      using namespace std;

      namespace AST {
        <method definitions 2b>
      }
```

```

21  <cec-grcsim.cpp 21>≡
    #include "GRCSim.hpp"
    #include <fstream>

    using namespace std;

    struct UsageError {};

    int main(int argc, char *argv[])
    {
        try {
            int debug = 0;
            int maxticks = -1;
            string testvectorfile;

            ++argv, --argc;

            while (argc > 0 && argv[0][0] == '-') {
                switch (argv[0][1]) {
                    case 'd':
                        if (argc == 1) throw UsageError();
                        ++argv, --argc;
                        debug = atoi(argv[0]);
                        break;
                    case 'c':
                        if (argc == 1) throw UsageError();
                        ++argv, --argc;
                        maxticks = atoi(argv[0]);
                        break;
                    case 't':
                        if (argc == 1) throw UsageError();
                        ++argv, --argc;
                        testvectorfile = argv[0];
                        break;
                    default:
                        cerr << "Unrecognized option " << argv[0] << endl;
                        throw UsageError();
                }
                ++argv, --argc;
            }

            if (argc != 0) throw UsageError();

            IR::XMLIstream r(std::cin);
            IR::Node *n;
            r >> n;

            AST::Modules *mods = dynamic_cast<AST::Modules*>(n);
            if (!mods) throw IR::Error("Root node is not a Modules object");

```

```

AST::Module *module = mods->modules.front();
assert(module);

AST::GRCgraph *gn = dynamic_cast<AST::GRCgraph*>(module->body);
assert(gn);

AST::GRCNode::NumMap cfgmap;
AST::STNode::NumMap stmap;

gn->enumerate(cfgmap, stmap);

AST::GRCsim simulator(gn, module, cfgmap, stmap, debug, std::cout);
if ( maxticks > 0 )
    simulator.execute_max(maxticks);
else if (!testvectorfile.empty()) {
    std::ifstream tvf(testvectorfile.c_str());
    if (tvf.bad()) {
        cerr << "Error opening test vector file " << testvectorfile << '\n';
        exit(-2);
    }
    simulator.execute_vectors(tvf);
    tvf.close();
} else throw UsageError();

} catch (IR::Error &e) {
    cerr << e.s << endl;
    exit(-1);
} catch (UsageError &) {
    cerr << "Usage: cec-grcsim [-d <level>] -c <cycles> | -t <vectorfile>\n"
        << "-d <level>    Enable debugging at the given level\n"
        << "-c <cycles>   Simulate with no inputs for this many cycles maximum\n"
        << "-t <vectorfile> Simulate taking inputs from the given vector file\n";
    exit(-1);
}

return 0;
}

```