SANS
Yosys signals analyzer module

Lorenzo CUGINI
Luca LAGNI

May 8, 2019

Date Performed: May 3, 2019
Professor: Professor Zoni Davide

1 Abstract

The project aims at analyzing input signals of a given Verilog specification.

The basic idea we've based our work upon is a construction of an hypergraph statring from the Verilog code.
At hyper-level each node represents a Verilog module whose connections to other modules mirror the high-level configuration of the circuit.
Inside each node we find another graph, this time nodes represent components like simple gates.
By running a DFS algorithm we're able to analyze each path a signal can take from its beginning.Every time the signal enters a selection port, we mark it as selection. Same reasoning undergoes data marking.
After the analysis has been performed, we convert the result in a readable format. By specifying options with command line, various files may be generated, including a .txt, .xml (you can also find a SAX parser in Java if you need to use the XML result), .json and .csv.

2 The pass module

The *pass.cc* is the files where we have defined the code of the ***sans*** command, the one used for building a graph of the design (i.e. the modules and their connections) passed to Yosys at run time and for analyzing its input signals , in order to decree if they can be categorized as *selection* and/or *data* signals.
When compiled and linked within Yosys, the ***sans*** command can be called.

Listing 1: bash command for compiling the pass.cc external module

```
#!/bin/bash
yosys-config --build pass.so pass.cc circuit.cc
```

The dependency of the *circuit.cc* is due to the fact that we have modified such file in order to manage the connection bits and, doing so, we are not using the default version in this case , so we have to add this file when we want to compile the *pass.cc*.

Listing 2: command for linking the previously compiled external module in Yosys

```
#!/bin/bash
yosys -m pass.so
```

Using the *sans* command

```
#!/bin/bash
      yosys-config --build pass.so pass.cc circuit.cc
      yosys -m pass.so -p "read_verilog file1.v...fileN.v
          ; proc; sans"
```

Part I
Basic Architecture

This section explains the basic elements on which the external module is build upon. They're used for represents RTLIL equivalent data but storing information that are useful to our purpose in a convenient way.

```
signal_library.h
port_library.h
node_library.h
generator_library.h
supportGraph.h
```

Note 1: in the following sections we have usually report also private attributes/methods in order to allow a better understanding of our design choices and make our solutions easier to read.

Note 2: We have listed all the public methods / functions but we have listed private methods, functions or attributes only when relevant.

3 Signal

Namespace that contains both a struct that represents a signal (called Signal) and methods used to operate over such structures or collections of them in a static way.

Accessibility:

File:	signal_library.h
Namespace:	Signal

Contents:

```
//Constants
EMPTYSIGNAL "empty"
//Signal structure
struct Signal{...};
//static methods
bool isIn(std::vector<Signal> haystack,Signal needle);
bool isIn_byNameOnly(std::vector<Signal> haystack,
    Signal needle);
std::vector<Signal> conditionalInsert_checkSignalName(
    std::vector<Signal> vector,Signal signal);
bool compare(const Signal& sig_1,const Signal& sig_2);
std::vector<Signal> sort(std::vector<Signal> vector);
Signal fetchSignal(std::vector<Signal> vec,std::signal
     sigName);
```

The different parts of the Signal namspace will be explained in the following sections.

3.1 Signal Strcuture

This structure has been used for representing signals that are exchanged between design components or with the external environment.
Since such physical signal can be used as a whole or only parts of them, they're representative of both RTLIL::Wire and RTLIL::Chunk elements (depending on their Yosys equivalent).
Since for our project only physical signals are relevat , every Signal instance will be associated to one of those signals (constants values which are associated to empty or unimportant values have not been taken into account).

Namespace:	Signal
External usage:	Signal::Signal
Yosys equivalent:	RTLIL::Chunk , RTLIL::Wire

3.1.1 Private Attributes

```
std::string signalName;
std::pair<int, int> dimensionPair;
```

signalName
label associated to the signal, equivalent to the one given in the corresponding verilog file.

dimensionPair
couple of int that repsenents the MSB (Most Significant Bit) and the LSB (Less Significant Bit) of the signal.

3.1.2 Public Constructors

```
Signal();
Signal(std::string signalName,int MSB,int LSB);
```

Signal(...)
constructor used only for temporary purposes **?**

Signal(std::string signalName,int MSB,int LSB)
constructor used for instantiating a Signal, if the signal is a single wire MSB = LSB.

3.1.3 Public Getter Methods

```
std::string getSignalName();
std::pair<int, int> getDimensionPair();
int getMSB();
int getLSB();
```

getSignalName(...)
getter for the signalName parameter.

getDimensionPair(...)
getter for the dimension pair couple (MSB, LSB).

getMSB(...)

getter for the Most Significant Bit associated to a Signal

getLSB(...)
getter for the Less Significant Bit associated to a Signal.

3.1.4 Public Utility Methods

```
bool isEmpty();
int width();
std::string toString();
```

isEmpty()
method used to decree if a signal is associated to some external signal.
Retruns true if the two signals are equivalent, false otherwise.

width()
method that returns the width of a signal (MSB - LSB + 1) as int.

toString()
mathod used for providing a human-readable representation of the signal in a string format

3.1.5 Operators

```
friend bool operator==(const Signal& sig_1,const Signal& sig_2);
```

operator==
comparison operator, returns true in case of two equivalent Signals (they're considered equivalent if they have the same name, same MSB and same LSB)

3.2 Constants

```
EMPTYSIGNAL "empty"
```

EMPTYSIGNAL
constant used to describe a port that is not used (i.e. it is not associated to any signal).

3.3 Static Methods

```
bool isIn( std :: vector <Signal> haystack,Signal needle);
bool isIn_byNameOnly(std::vector<Signal> haystack, Signal needle);
std :: vector <Signal> conditionalInsert_checkSignalName(std :: vector <Signal> vector,Signal signal);
bool compare(const Signal& sig_1,const Signal& sig_2);
std :: vector <Signal> sort(std:: vector <Signal> vector);
Signal   fetchSignal ( std :: vector <Signal> vec,std::signal  sigName);
```

isIn(...)
check if the (needle) signal is in the passed (haystack) list, based on name, MSB and LSB equivalence

isIn_byNameOnly(...)
like isIn but checks only on signal name equivalence.

conditionalInsert_checkSignalName(...)
method used for inserting a new Signal into a list iff the list doesn't have any signals with the same name already in.

compare(...)
comparison between two signals based on their name, MSB and LSB values.

sort(...)
method used for sorting a vector of signals

fetchSignal(...)
method used for retriving a signal from a list based on its name

4 Port

Namespace that contains several components used to represent (and operate) the equivalent of pins for a component.
Stricly speaking, the Port struct is equivalent to an RTLIL::Cell.

Accessibility:

File:	port_library.h
Namespace:	Graph_Port

Contents:

```
//Enums
enum PortDirection{...}
//Port
struct Port{...};
```

The different parts of the Port namspace will be explained in the following sections.

4.1 Enum

The purpose of this enum is to define a fixed number of possible values associated to the direction of a port.

```
INVALID;
INPUT;
OUTPUT;
INOUT;
```

INVALID
label associated to a port that has an unknown port direction which , in our case, it's a mistaken

INPUT
label associated to an input port

OUTPUT
label associated to an output port

INOUT

label associated to an input-output port

4.2 Port Strcuture

This structure has been used for representing post (i.e. pins) associated to a circuit component.
Their Yosys equivalent are the RTLIL::Cell elements.
Ports are the endpoints for signals when they're defined as communications paths between nodes.

External usage:	Graph_Port::Port
Yosys equivalent:	RTLIL::Cell

4.2.1 Private Attributes

```
std::string portName;
int portId;
PortDirection direction;
pool<int> edges;
std::vector<Signal::Signal> inputSignals;
std::vector<Signal::Signal> outputSignals;
bool connectedToExtern;
```

Here it's easy to see that inputSignals and outputSignals are declined as vectors, this means that a port can have associated a single signal (direct wire) or multiple signals (bus).

portName
label associated to the port

portId
integer value used for idenify the port associated to a specific component (it will be used for analyzing connection between nodes).

direction
type signals direction associated to a specific port

edges
set of references of other ports that are directly connected to this one

inputSignals
set of signals that enters in the port in case of input or inout port

outputSignals
set of signals that exit from the port in case of output or inout port

connectedToExtern
boolean flag used to decree if a port is associated to an output of a module.
It is only used in case of output or inout ports

4.2.2 Public Methods

This section lists all the methods available for the Port struct.
Their meaning and usage will be explained later when needed.

```
public:
        //Constructors
        Port();
        Port(std::string portName);
        Port(std::string portName,int portId);
        Port(std::string portName,PortDirection direction);
        Port(std::string portName,PortDirection direction, int portID);
        //Canonical getters
        std::string getPortName();
        int getPortId();
        PortDirection getDirection();
        pool<int> getEdges();
        std::vector<Signal::Signal> getInputSignals();
        std::vector<Signal::Signal> getOutputSignals();
        bool isConnectedToExtern();
        //Non-canonical getters
        std::vector<Signal::Signal> getSignals();
        //Setters
        void setConnectionToExtern(bool conn);
        void setDirection(PortDirection dir);
        //Booleans
        bool isValidPortDirection();
        bool isInputPort();
        bool isOutputPort();
        bool isInOutPort();
        bool isInvalidPort();
        //Adders
        void pushInputSignals(Signal::Signal sig);
        void pushOutputSignals(Signal::Signal sig);
        void pushInOutSignal(Signal::Signal sig);
        void addEdge(int edgeId);
        //toString();
        std::string toString();
```

4.2.3 Public Constructors

In this section we are going to only lists constructors methods because we think that the previous explanation that we have done before for associated attributes can already defines their purposes.

```
Port();
Port(std::string portName);
Port(std::string portName,int portId);
Port(std::string portName,PortDirection direction);
Port(std::string portName,PortDirection direction, int portID);
```

note: Here we have several different constructors because we can set in a further moments the parameters for a specific port.

In particular, it's in the graph build phase that we differ the setup of a port parameters.

4.2.4 Canonical Getter Methods

In this section we are going to only lists the getter methods for the associated attributes , since we think that the previous explanation that we have done before for such attributes can already defines their purposes.

```
std::string getPortName();
int getPortId();
PortDirection getPortDirection();
pool<int> getEdges();
std::vector<Signal::Signal> getInputSignals();
std::vector<Signal::SIgnal> getOutputSignals();
bool isConnectedToExtern();
```

4.2.5 Non-canonical Getter Methods

This getter methods are called non canonical in the sens that they're not associated to class/strcut attributes but rather on groups or manipulations perfomed on such attributes.
Their meaning and usage will be explained in the following section.

```
std::vector<Signal::Signal> getSignals();
```

getSignals()
Method that returns all the Signals associated to a port , that can be Input, Output or Inout.

4.2.6 Setter Methods

These setter methods are used for setting attributes of the port.
Their meaning will be explained only when required since we think that the previous explanation about the struct attributes has explained enough their corresponding purposes.

```
void setConnectionToExtern(bool conn);
void setDirection(PortDirection direction);
```

4.2.7 Boolean methods

This methods are used to get information about a specific state of a Port object.
We are not going to explain them since we think they're enougth self explanatory , based on the corresponding attributes.

```
//Booleans
bool isValidPortDirection();
bool isInputPort();
bool isOutputPort();
bool isInOutPort();
bool isInvalidPort();
```

4.2.8 Public Adders methods

These methods are used for adding a specific element to a Port.

```
//Used for adding an input Signal to the Port.
void pushInputSignals(Signal::Signal sig);
//Used for adding an output Signal to the port
void pushOutputSignals(Signal::Signal sig);
//Used for adding an InOut Signal the port.
void pushInOutSignal(Signal::Signal sig);
//Used for adding an edge to the port
void addEdge(int edgeId);
```

pushInOutSignal(Signal::Signal sig)
Since we don't have a list for InOut signals (because it would have been redundant), when we push a new InOut signal, we are simply adding such signal to both the inputSignals and outputSignals vectors.

4.2.9 toString() method

```
//toString();
std::string toString();
```

5 Node

Namespace that contains several the structure for representing a node (i.e. a component of the circuit).
Nodes are equivalent to both atomic components like mux, adders etc. (RTLIL::Cell) and modules (RTLIL::Module), depending on the nature of associated components.

Accessibility:

File:	node_library.h
Namespace:	Graph_Node

Contents:

```
//Struct
struct Node{...};
```

5.1 Node Strcuture

This structure has been used for representing node (i.e. component) associated to a circuit.
Their Yosys equivalent are the RTLIL::Cell which in some cases are only placeholders for RTLIL::Modules.

External usage:	Graph_Node::Node
Yosys equivalent:	RTLIL::Cell

5.1.1 Private Attributes

```
private:
      int nodeID;
      std::string name;
      std::string type;
      bool modulePlaceholder ;
      std::vector<Graph_Port::Port *> inputPorts
         ;
      std::vector<Graph_Port::Port *>
         outputPorts;
      std::vector<Graph_Port::Port *> inoutPorts
         ;
```

nodeID
Unique numeric identifier of the node.

name
name associated to a node; it represents how it is defined when it is instantiated (in case of modules) or how it is defined when translated in the RTL for by Yosys (in case of atomic components).

type
string that identify the type of a component, i.e. which element of a circuit it represents (mux, and-port,module ...).

modulePlaceholder
boolean flag setted to true in case of node representing a Module (modules are threated in a different ways since they must be inspected in their internal representation).

inputPorts
vectors of input ports of a node.

outputPorts
vectors of output ports of a node.

inoutPorts
vectors of inout ports of a node.

5.1.2 Public Methods

This section lists all the methods available for the Node struct.
Their meaning and usage will be explained later when needed.

```
public:
        //Constructor
        Node(int nodeID,std:: string  nodeName,std:: string  nodeType,bool modulePlaceholder);
        //Canonical Getters
        int getId();
        std :: string  getName();
        std :: string  getType();
        //Non-Canonical Getters
        std :: vector <Graph_Port::Port * > getInputs();
        std :: vector <Graph_Port::Port * > getOutputs();
        std :: vector <Graph_Port::Port * > getPorts();
        Graph_Port:: Port  *getPort(int id);
        std :: vector <Signal::Signal > getInputSignals();
        std :: vector <Signal::Signal > getOutputSignals();
        //Booleans
        bool isEmitter();
```

```
bool isModulePlaceholder();
//Adders
int addPort(Graph_Port:: Port *newPort,Graph_Port:: Direction direction );
//toString()
std :: string toString ();
```

5.1.3 Public Constructors

```
Node(int nodeID,std::string nodeName,std::string
    nodeType,bool modulePlaceholder);
```

Note: It's important to notice that in this case we have only one constructor, this is due to the fact that we cannot have, for example , an empty Node, since each Node correspond to a specific element that must be present in the circuit.

5.1.4 Canonical Getter Methods

```
int getId();
std::string getName();
std::string getType();
```

5.1.5 Non-canonical Getter Methods

This getter methods are called non canonical in the sens that they're not associated to class/strcut attributes but rather on groups or manipulations performed on such attributes.
Their meaning and usage will be explained in the following section when needed.

```
std::vector<Graph_Port::Port *> getInputs();
std::vector<Graph_Port::Port *> getOutputs();
std::vector<Graph_Port::Port *> getPorts();
Graph_Port::Port *getPort(int id);
std::vector<Signal::Signal> getInputSignals();
std::vector<Signal::Signal> getOutputSignals();
```

getInputs()
Method that returns a vector that contains all the input Ports associated to a node.
Here we have both Input Ports and InOut Ports.

getOutputs()
Method that returns a vector that contains all the output Ports associated to a node. Here we have both Output Ports and InOut Ports.

getPorts()
Method used for getting all the Port associated to a Node (Input, Output and InOut Ports).

***getPort(int id)**
Method that return a reference to a specific port, based on the passed id, in case such port is contained in ,at least, one of the associated vectors, return nullptr otherwise.

getInputSignals()
method that return all the input Signals that are associated to a node.

getOutputSignals()
method that return all the output Signals that are associated to a node.

5.1.6 Booleans Methods

```
bool isEmitter();
bool isModulePlaceholder();
```

isEmitter()
return true if at least one one of its outputs (output/inout ports) is connected to extern.

isModulePlaceholder()
return true in case of a node that is a placeholder for a module (i.e. in the current node is instantiated another module), false if the corresponding node is an atomic component.

5.1.7 Adder Methods

```
int addPort(Graph_Port::Port *newPort,Graph_Port::Direction direction);
```

addPort(Graph_Port::Port *newPort,Graph_Port::Direction direction)
method used for adding a specific Port to the current node.

It can returns different integer values, depending on the following situations:

-1	the passed port is already present as Input Port
-2	the passed port is already present as Output Port
-3	the passed port is already present as InOut Port
-4	the passed port as an unknown direction
1	the passed port has been added in the corresponding vector

5.1.8 ToString method

```
std::string toString();
```

6 SupportGraph

SupportGraph is both the name of the namespace and the class associated to this element of the design.

Accessibility:

File:	supportGraph.h
Namespace:	SupportGraph

Contents:

```
//Class
class SupportGraph{...};
```

6.1 SupportGraph Class

The SupportGraph class represents an entire module of the design: components and connections of elements of the RTLIL::Design.

External usage:	SupportGraph::SupportGraph
Yosys equivalent:	RTLIL::Module

6.1.1 Private Attributes

```
private:
      CellRepository::CellRepository *
         cellrepository;
      std::vector<Graph_Node::Node *> nodes;
      int numNodes;
```

cellrepository
reference to an instance of CellRepository , which is a struct used for knowing the different aspects regarding the ports of a node (explained in a farther section).

nodes
collections of all the nodes that are contained in a specific module.

numNodes
number of nodes of a specific module

6.1.2 Public Methods

This section lists all the methods available for the SupportGraph class.
Their meaning and usage will be explained later when needed.

```
public:
//Canonical getter methods
CellRepository :: CellRepository  *getCellRepository () ;
std :: vector <Graph_Node::Node *>getNodes();
int getNumNodes();
//Non canonical getter methods
Graph_Node::Node *getNode(int id);
//Booleans
bool isNodeName(std::string name);
//Adders
int addNode(Graph_Node::Node *newNode);
//Utilities
std :: vector <Graph_Node::Node *> nextNodes(Graph_Node::Node *node,Graph_Port::Port *port,bool discriminating,bool verbose);
std :: vector <Graph_Node::Node *> allNextNodes(Graph_Node::Node *node,bool discriminating,bool verbose);
std :: vector <std::pair <Graph_Port::Port *, int>> reachedPorts(Graph_Node::Node *sourceNode,Graph_Port::Port *sourcePort,Graph_Node::Node *
        destNode);
//ToString
std :: string  toString () ;
```

Note: here we don't have any explicit constructor since this class corresponds to a collection of Node-types classes.

6.1.3 Canonical Getter Methods

```
CellRepository::CellRepository *
    getCellRepository();
std::vector<Graph_Node::Node *>getNodes();
int getNumNodes();
```

6.1.4 Non-canonical Getter Methods

```
Graph_Node::Node *getNode(int id);
```

***getNode(...)**
method that returns a specific node based on the nodeid passed as parameter (null-pointer otherwise).

6.1.5 Boolean Methods

```
bool isNodeName(std::string name);
```

isNodeName(...)
checks if the passed name is the one of an existing node.

6.1.6 Adder Methods

```
int addNode(Graph_Node::Node *newNode);
```

addNode(...)
method used for adding a specific Node to the current support graph

It can returns different integer values, depending on the following situations:

-1	we already have a node with the same name in the current SupportGraph instance
1	we have correctly added the new node to the current SupportGraph instance

6.1.7 Utilities Methods

```
std :: vector <Graph_Node::Node *> nextNodes(Graph_Node::Node *node,Graph_Port::Port *port,bool discriminating,bool verbose);
std :: vector <Graph_Node::Node *> allNextNodes(Graph_Node::Node *node,bool discriminating,bool verbose);
std :: vector <std::pair <Graph_Port::Port *, int>> reachedPorts(Graph_Node::Node *sourceNode,Graph_Port::Port *sourcePort,Graph_Node::Node *
    destNode);
```

nextNodes(...)
method that a vector of nodes that can be reached by the parameter node, passing through the port passed as parameter.
The discriminating boolean value is used to define if we want (false) to consider as canonical-data input also the selection ports or not(true).
The verbose boolean flag is used for stamping some values during each passage in case it is setted as true.

allNextNodes(...)
method that a vector of nodes that can be reached by the parameter node, by any of its output ports
The discriminating boolean value is used to define if we want (false) to consider as canonical-data input also the selection ports or not(true).
The verbose boolean flag is used for stamping some values during each passage in case it is setted as true.

reachedPorts(...)
method that returns a vector of pairs used to represents which ports of the destination nodes can be reached by the sourcePort of the sourceNode

6.1.8 ToString method

```
std::string toString();
```

7 Generator

This namespace contains all the functions that we have implemented for building our custom structure starting from the RTLIL::Design.
The associated structure (Generator) is used for representing a module.

Accessibility:

File:	generator_library.h
Namespace:	Generator

Contents:

```
//Constants
NONE 'N'
SELECT 'S'
DATA 'D'
BOTH 'B'
ERROR_CODE 'E'
//Static functions
char nextLetter(char currentNode,std::pair<bool,
    bool> data_select);
//Class
struct Generator : SubCircuit_v2::Graph{...}
```

7.1 Constants

These constants has been used for defining the type of signal associated to each bit of a Signal.

```
//The signal bit is not used
NONE 'N'
//The signal bit is used as selection
SELECT 'S'
//The signal bit is used as data
DATA 'D'
//The signal bit is used both as data and
   selection
BOTH 'B'
//Invalid value
ERROR_CODE 'E'
```

7.2 Static functions

```
char nextLetter(char currentNode,std::pair<bool,
    bool> data_select);
```

nextLetter(...)
Static utility function that decides what letter the support structure shall contains for each bit of each signal.

7.3 Struct Generator

This structure has been used for representing a module of the design but is not only that, it is a top-level container used for performing also Operations on such module.

External usage:	Generator::Generator
Yosys equivalent:	RTLIL::Module

7.3.1 Private Attributes

```
private:
std :: string  moduleName;
SupportGraph::SupportGraph sg;
std :: vector <Signal::Signal> externalSignals;
std :: vector <Signal::Signal> enteringSignals;
std :: vector <Signal::Signal> exitingSignals;
std :: vector <Signal::Signal> inoutSignals;
std :: vector <std::pair <std:: string , std :: string > > coupledSignals;
std :: vector <std::pair <Signal::Signal, Signal :: Signal > > pairedSignals;
std :: vector <std::pair <std:: string , std :: string > > innerModules;
std :: vector <std::pair <Signal::Signal,  std :: vector <char > > > sliceStructure;
```

moduleName
name of the module that Generator wrap.

sg
Internal structure of a single module.

externalSignals
vector of external signals of the module.

enteringSignals
Vector of signals entering the module

exitingSignals
vector of signals exiting the module

inoutSignals
Vector of inout signals of the module

coupledSignals
Support structure to fill the exitingSignals even in case of signal renaming.

pairedSignals
vector that contains all the infos related to coupled signals.

innerModules
Set of modules that are instantiated inside the current module (¡name , type¿), used to decree if a module is atomic or not.

sliceStructure
Support for bit condensing.

7.3.2 Private Methods

This section lists and explains the functions of each private method.
They're not directly accessible but they're very important for our application since they're responsable for building the entire application architecture.

```
private:
        void  fillSignalStructure  () ;
        void extractInnerModules() ;
        std :: pair <std::vector <Signal::Signal>,std:: vector <Signal::Signal> > extractModuleOutputBinding(RTLIL::Module *module);
        void setCellModuleOutputBindings(RTLIL::Cell *cell,Graph_Port :: Port  *newPort,std :: pair <std::vector <Signal::Signal>, std :: vector <Signal::
                Signal> > moduleOutputBindings,bool inout);
        bool buildGraph(SubCircuit_v2 :: Graph \& myg, RTLIL::Module *module);
        bool buildConsistentGraph(SubCircuit_v2 :: Graph \& myg,RTLIL::Module *module,bool verbose=false);
        void prepareSlices () ;
```

fillSignalStructure(...)
Method that corrects the content of externalSignals and co.

extractInnerModules(...)
Method used for automatically extract inner modules from internal structure.

extractModuleOutputBinding(...)
Method used for extracting the connections between internal cells outputs and the module's outputs.

setCellModuleOutputBindings(...)
Method used for knowing if a specific cell's output has some bindings with the module's outputs

buildGraph(...)
Method that generates the final graph of a module by taking the default subcircuit graph and the design.

buildConsistentGraph(...)
Method that builds the final graph and then check it's consistency.

prepareSlices(...)
Method that returns the structure we need to perform bit condensing.

7.3.3 Protected Methods

```
protected:
//Check that no port has direction INVALID_PORT
int CHECK_portIsValid(bool verbose=false);
//Check portId validity
int CHECK_portIDValid(bool verbose=false);
//Ports connected to extern must be outputs or inouts
int CHECK_portConnectionToExtern(bool verbose=false);
//Check that nodes where counted correctly
int CHECK_correctNumOfNodes(bool verbose=false);
//Checking that inputs are not set on output ports and viceversa
int CHECK_consistentSignalVectors(bool verbose=false);
//Method used to check if results of startingNodes and endingNodes matches with Receiver and Emitter
      nodes
int CHECK_SE_match_RE(bool verbose=false);
//Check slice structure's consistency
int CHECK_StructureSupport(bool verbose=false);
//After we have build the graph, we need to check if everything was done correctly. This method does
      so.
bool check(bool verbose=false);
```

7.3.4 Public Methods

```
public:
      //Constructor
      Generator(SubCircuit_v2::Graph& myg, RTLIL::Module *module, bool verbose=false);
      //Setters
      void setSliceStructure ( std::vector< std::pair< Signal::Signal, std::vector<char> > > structure );
      //Core methods
      std::vector<Signal::Signal> findSignalFriends(Signal::Signal sig, bool includeSelf);
      std::vector<Graph_Node::Node *> startingNodes_fullSignal( Signal::Signal externalSignal, bool discriminating, bool verbose=false);
```

```
std :: vector <Graph_Node::Node *> startingNodes_byNameOnly( Signal::Signal externalSignal, bool discriminating, bool verbose=false );
std :: string  getRenaming(std :: string  signalName, bool getNew=true);
bool isExternalSignal_byNameOnly(Signal::Signal  signal );
std :: vector <Graph_Node::Node *> signalReceivers(bool discriminating);
std :: vector <Graph_Node::Node *> signalEmitters();
bool isReceiver(Graph_Node::Node *node);
std :: string  condenseBit ();
bool isAtomic();
//Getters
std :: string  getModuleName(){ return this->moduleName; }
SupportGraph::SupportGraph getSupportGraph();
std :: vector <Signal::Signal> getEnteringSignals ();
std :: vector <Signal::Signal> getExitingSignals ();
std :: vector <Signal::Signal> getExternalSignals ();
std :: vector <std::pair < Signal::Signal, std :: vector <char> > > getSliceStructure();
std :: vector <std::pair <std:: string , std :: string > > getCoupledSignals();
std :: vector <std::pair <Signal::Signal, Signal :: Signal > > getPairedSignals();
std :: vector <std::pair <std:: string , std :: string > > getInnerModules();
std :: vector <Signal::Signal> getInoutSignals ();
//toString()
std :: string  toString ();
```

7.3.5 Public Constructors

```
Generator(SubCircuit_v2::Graph& myg, RTLIL::
    Module *module, bool verbose=false);
```

Generator(...)
Method used for building a new Generator starting from an RTLIL::Module.
The SubcircuitGraph is passed as a reference for the internal structure (to be setted) an the verbose boolean flag is used for define if we have to show passages during the building of the Generator

7.3.6 Setter Method

```
void setSliceStructure( std::vector< std::pair<
    Signal::Signal, std::vector<char> > >
    structure );
```

setSliceStructure(...)
Method used for preparing the slice structure for the current Generator

7.3.7 Core Methods

```
std::vector<Signal::Signal> findSignalFriends(
    Signal::Signal sig, bool includeSelf);
std::vector<Graph_Node::Node *>
```

```
        startingNodes_fullSignal( Signal::Signal
        externalSignal, bool discriminating, bool
        verbose=false);
    std::vector<Graph_Node::Node *>
        startingNodes_byNameOnly( Signal::Signal
        externalSignal, bool discriminating, bool
        verbose=false );
    std::string getRenaming(std::string signalName,
        bool getNew=true);
    bool isExternalSignal_byNameOnly(Signal::Signal
        signal);
    std::vector<Graph_Node::Node *> signalReceivers(
        bool discriminating);
    std::vector<Graph_Node::Node *> signalEmitters()
        ;
    bool isReceiver(Graph_Node::Node *node);
    std::string condenseBit();
    bool isAtomic();
```

findSignalFriends(...)
Method that , given a signal,returns all the slices sig[x:y] which appear as inputs in some ports.

startingNodes_fullSignal(...)
ethod that, given the name of a signal, returns a list of nodes where said signal enters as a pure signal (meaning without any modification whatsoever).

startingNodes_byNameOnly(...)
Method that, given the name of a signal, returns a list of nodes where said signal enters as a pure signal (meaning without any modification whatsoever).

getRenaming(...)
Method that finds a correspondant among the coupledSignals.
If getNew, we search for a forward renaming y such that x -¿ y, otherwise x ¡- y ¡new-Name , oldName¿

isExternalSignal_byNameOnly(...)
Method that checks if the signal passed is an external signal

signalReceivers(...)

Method that returns the list of nodes which receives as inputs signals coming from outside the current module.

signalEmitters(...)
Method that returns the list of nodes which outputs a signal exiting the current module.

isReceiver(...)
Method used to decree if the current node is a receiver or not

condenseBit(...)
Method to transform the support structure in a string.

isAtomic(...)
Method used to decide if a module contains other modules (return true) or not(return false)

7.3.8 Getter Methods

Since these are simple getter methods for the corresponding attributes , we think that their meaning and returned value is enough self-explanatory, based on what we have said before about the corresponding attributes.

```
std::string getModuleName(){ return this->
    moduleName; }
SupportGraph::SupportGraph getSupportGraph();
std::vector<Signal::Signal> getEnteringSignals()
    ;
std::vector<Signal::Signal> getExitingSignals();
std::vector<Signal::Signal> getExternalSignals()
    ;
std::vector<std::pair< Signal::Signal, std::
    vector<char> > > getSliceStructure();
std::vector<std::pair<std::string, std::string>
    > getCoupledSignals();
std::vector<std::pair<Signal::Signal, Signal::
    Signal>> getPairedSignals();
std::vector<std::pair<std::string,std::string>>
    getInnerModules();
```

```
        std::vector<Signal::Signal> getInoutSignals();
```

7.3.9 ToString()

```
        std::string toString();
```

Part II
Analyzers

This section contains the structures used for analyzing the design that we have dumped from the RTLIL::Design and rebuilded in our manner for our application purposes.

```
        //files
        generator_analyzer.h
        dfs.h
        design_analyzer.h
```

This part differs from the previous one , since here we don't have any direct correspondence between our solution and the design provided by Yosys.

8 GeneratorAnalyzer

This namespace contains utilities functions used for analyzing several different aspects of the Generator structure.

The content of this file is a data structure , GeneratorAnalyzer, used for testing some characteristics of the Generator class.

Accessibility:

File:	generator_analyzer.h
Namespace:	GeneratorAnalyzer

Contents:

```
//Struct
struct GeneratorAnalyzer{...}
```

Note: not all the private methods and attributes will be explained here, only the relevant ones.

8.1 GeneratorAnalyzer struct

This struct is used for testing several aspects of a Generator that we are going to explain in the next section.

External usage: GeneratorAnalyzer::GeneratorAnalyzer
Note: this struct doesn't contains any attribute since it works on passed Generator instances.

8.1.1 Public Methods

This section lists all the methods available for the GeneratorAnalyzer struct.

```
public:
    //Constructors
    GeneratorAnalyzer();
    //Testers
    void TEST_findSignalFriends(Generator::Generator *wrappedModule);
    void TEST_node_classification(Generator::Generator *wrappedModule);
    void TEST_entering_emitter_nodes(Generator::Generator *wrappedModule);
    void TEST_startingNodes(Generator::Generator *wrappedModule, bool verbose, bool checkAllSignals=false);
    void TEST_all(Generator::Generator *wrappedModule);
    //Printers
    void PRINT_externalSignals(Generator::Generator *wrappedModule):
    void PRINT_supportGraph(Generator::Generator *wrappedModule);
    void PRINT_all(Generator::Generator *wrappedModule);
```

We have such division between testers and printers since testers performs some kind of discrimination over certain attributes of a Generator , while printers simply will print all the elements that are part of a specific aspect of a Generator, regardless of their function or usage.

8.1.2 Public Constructors

```
//Method used for instantiating a
   GeneratorAnalyzer
GeneratorAnalyzer();
```

8.1.3 Testers

```
void TEST_findSignalFriends(Generator::Generator *wrappedModule);
void TEST_node_classification(Generator::Generator *wrappedModule);
void TEST_entering_emitter_nodes(Generator::Generator *wrappedModule);
void TEST_startingNodes(Generator::Generator *wrappedModule, bool verbose, bool checkAllSignals);
void TEST_all(Generator::Generator *wrappedModule);
```

TEST_findSignalFriends(...)
Method used for exploring all the signal friends that are present in a Generator.
In will print a list of nodes that are friends and return nothing .

TEST_node_classification(...)
Method that prints which nodes of a generator are Emitters (node connected to , at last, one output of the module) and which nodes are Receivers (node connected to , at last, one input of the module)

TEST_entering_emitter_nodes(...)
Method that prints, for every module's level signals, which nodes are Emitters or Receivers for such signal.

TEST_startingNodes(...)
Method used for printing which nodes are starting nodes for module level's signals.

TEST_all(...)
Method that calls all the methods above and , so , it will list all the infos printed by such methods

8.1.4 Printers

```
void PRINT_externalSignals(Generator::Generator *wrappedModule);
void PRINT_supportGraph(Generator::Generator *wrappedModule);
void PRINT_all(Generator::Generator *wrappedModule);
```

PRINT_externalSignals(...)
Method that prints on screen all the external signals of a module.

PRINT_supportGraph(...)
Method that prints the support graph (internal structure) of a module.

PRINT_all(...)
Method that calls all the method above, printing all the infos printed by such method but with a single function call.

9 DFS

This file contains all the data structures and methods used for analyzing signals and decree if a signal is selection , data one , both or none.

Accessibility:

File: dfs.h
Namespace: DFS

Contents:

```
//Classes
class DFSDSS{...}
class DFS{...}
```

Note: not all the private methods and attributes will be explained here, only the relevant ones.

9.1 DFSDSS class

The purpose of this class is to speed-up the execution of the deep-first scan so, we are not going to explain in details its functionalities since they're useful only fo the DFS class.

External usage: DFS::DFSDSS

```
private:
    //Reference to the module we're analyzing
    Generator::Generator *targetModule;
    //Signals that we have already analyzed along the result <data=bool,select=bool>
    std::vector<std::pair<Signal::Signal, std::pair<bool,bool>>> previouslyProcessedResults;
public:
    //Constructors
    DFSDSS(Generator::Generator *targetModule);
    //Setters
    void setPreviouslyProcessedResults(std::vector<std::pair<Signal::Signal, std::pair<bool,bool>>>
        previouslyProcessedResults);
    //Getters
    Generator::Generator *getTargetModule();
    std::vector<std::pair<Signal::Signal, std::pair<bool,bool>>> getPreviouslyProcessedResults();
```

9.2 DFS class

Class used for perfmoing the DFS , in order to decree if a signal is a data, selection , both or none of the mentioned cases.

External usage: DFS::DFS

9.2.1 Private Attributes

```
private:
      //Reference of the module that we are
         analyzing
      Generator::Generator *currentModule;
      //List of all the (processed) modules of
         the design
      std::vector<Generator::Generator *>
         designModules;
      //Reference to a cell repository instance
      CellRepository::CellRepository *cr;
      bool deepScanEnabled;
```

deepScanEnabled
Boolean flag used for enabling or disabling the deep scan mode which means looking inside the internal structure of modules that are contained in the currentModule in order to decide the type of signal that are passed to the inner module.

9.2.2 Private Methods

```
private:
      bool DFS_MooreCorrection( Signal::Signal
         signal );
      void DFS_Data( Graph_Node::Node *
         currentNode, std::vector<bool> visited
         , bool &data );
      void DFS_Select( Graph_Node::Node *
         currentNode, std::vector<bool> visited
         , bool &select );
      bool DFS_DeepScan( Graph_Node::Node *
         analyzedNode,Graph_Port::Port *
         analyzedPort );
```

DFS_MooreCorrection(...)
Method that covers the last possible selection case that the DFS could not cover due to forced discrimination.
If the signal enters straight into a selection port, discrimination will trim the whole node before the analysis could proceed.

DFS_Data(...)
Method that performs the checks to know whether a signal is a data signal. The result is stored inside bool& data.

DFS_Select(...)
Method that performs the checks to know whether a signal is a selection signal. The result is stored inside bool& select.

DFS_DeepScan(...)
Method used for looking inside inner modules in order to decide if a signal is used as DATA/SELECT by inner modules.

9.2.3 Protected Methods

This section contains the methods that are used for testing functionalities of this class over the currentModule passed.
They has been used for debugging purpose and has been kept , since we think that they're output can be meaningful or, at last , interesting.
Since they're quite easy to understand and use , we are not going to describe them in detail.

```
protected:
      //Method that prints the result of the DFS over all the current module's nodes.
      void TEST_DFS_NodeAnalyzer();
      //Method that prints the result of the DFS over all the current module's external signals.
      void TEST_DFS_SignalAnalyzer();
```

9.2.4 Public Methods

```
public:
//Constructors
DFS( Generator::Generator *currentModule, std::vector<Generator::Generator*> designModules, bool deepScanEnabled = true );
//Setters
void setPreviouslyProcessedModules(std::vector<DFSDSS*> previouslyProcessedModules);
//Getters
Generator::Generator *getCurrentModule();
std::vector<Generator::Generator*> getDesignModules();
bool isDeepScanEnabled();
//Core methods
std::pair<bool,bool> DFS_SignalAnalyzer( Signal::Signal signal, bool verbose=false );
std::pair<bool,bool> DFS_NodeAnalyzer( Graph_Node::Node *currentNode, bool verbose=false );
std::vector< std::pair< Signal::Signal , std::pair<bool,bool> > > DFS_FTSA( bool verbose=false );
//Utilities
bool enableDeepScan();
bool disableDeepScan();
```

9.2.5 Constructor

```
DFS( Generator::Generator *currentModule, std::
    vector<Generator::Generator *> designModules
    , bool deepScanEnabled = true );
```

DFS(...)
This constructor requires the currentModule to analyze, the list of other modules of the design (wrapped as Generators) in order to allow the deepscan to be used in case of inner modules and a boolean used to define if we can perform the deepscan.

9.2.6 Public Setters

```
public:
    //Method used for passing previously processed
        signals to the current one, used for
        optimizing the performances of the deep scan
         in successive modules DFS call.
    void setPreviouslyProcessedModules(std::vector<
        DFSDSS *> previouslyProcessedModules);
```

9.2.7 Public Getters

```
public:
    //Module on which we are operating the DFS
    Generator::Generator *getCurrentModule();
    //List of all the wrapped modules obtained from
        the design
    std::vector<Generator::Generator *>
        getDesignModules();
    //Used for knowing if the deepscan is enabled or
         not
    bool isDeepScanEnabled();
```

9.2.8 Core Methods

This methods are the ones that can be called from the outside (and performs the DFS on specific elements of the currentModule) and provides the DFS analysis in order to decree if signals coming and going from nodes or modules are selection, data or both.

```
public:
        std :: pair <bool, bool> DFS_SignalAnalyzer( Signal::Signal signal, bool verbose=false );
        std :: pair <bool, bool> DFS_NodeAnalyzer( Graph_Node::Node *currentNode, bool verbose=false );
        std :: vector < std:: pair < Signal::Signal , std :: pair <bool, bool> > > DFS_FTSA( bool verbose=false );
```

DFS_SignalAnalyzer(...)
Given a single signal it returns the result of a DFS Select + Data analysis in order to decree if such signal has been used as data or selection.

DFS_NodeAnalyzer(...)
Method that prepares the field for the DFS algorithm to explore paths.
The returned pair has the shape ¡Data, Select¿ .

DFS_FTSA(...)
DFS analysis with bit slices enabled.
This methods analyze all the entering signals of a module and produces , for each of them , the couple that indicates if a signal is a data or selection signal

10 DesignAnalyzer

This namespace contains the definition for the data structure used for analyzing the RTLIL::Design which modules has been wrapped into Generators.

Accessibility:

File:	design_analyzer.h
Namespace:	DesignAnalyzer

Contents:

```
//Structs
struct Final_Results{...};
struct DesignAnalyzer{...};
```

10.1 Final_Results

This data structure is used for instantiating a single global variable (dfs_results) that contains all the results in a user-friendly format that will be further used for storing such results in a persistent format.

External usage: DesignAnalyzer::Final_Results

```
std::string plaintext;
std::string xml;
std::string json;
std::string csv;
```

plaintext
In this string is contained the results that will be written in the txt file.

xml
In this string is contained the results that will be written in the xml file.

json
In this string is contained the results that will be written in the json file.

csv
In this string is contained the results that will be written in the csv file.

It's important to notice that the corresponding declaration doesn't correspond to a data type definition but it is only used for instantiating the global variable defined above.//

10.2 DesignAnalyzer struct

This struct is used for instantiating objects that will be used for analyzing the RTLIL::Design (once it will be converted in our internal format).

External usage: DesignAnalyzer::DesignAnalyzer

10.2.1 Private Attributes

```
//List of all the modules of the RTLIL::Design converted in our internal format
std :: vector <Generator::Generator *> wrappedModules
```

10.2.2 Private Methods

```
private:
        void wrapModule(RTLIL::Module *module);
        std :: vector < std:: pair < Signal:: Signal , std :: pair <bool, bool> > > sort( std::vector< std::pair< Signal::Signal, std:: pair <
                bool, bool> > > toSort);
        std :: vector <std:: string > prepareResults( std :: string title , Generator :: Generator *mod, std :: vector < std:: pair < Signal:: Signal ,
                std :: pair <bool, bool> > > allResults, bool condense=false );
        std :: string preparePlaintext ( std :: string title , Generator :: Generator *mod, std :: vector < std:: pair < Signal:: Signal , std :: pair <
                bool, bool> > > allResults, bool condense=false );
        std :: string prepareXML( std:: string title , Generator :: Generator *mod, std :: vector < std:: pair < Signal:: Signal , std :: pair <bool,
                bool> > > allResults, bool condense=false );
        std :: string prepareJSON( std :: string title , Generator :: Generator *mod, std :: vector < std:: pair < Signal:: Signal , std :: pair <bool,
                 bool> > > allResults, bool condense=false );
        std :: string prepareCSV( std :: string title , Generator :: Generator *mod, std :: vector < std:: pair < Signal:: Signal , std :: pair <bool,
                bool> > > allResults, bool condense=false );
```

wrapModule(...)
Method used for creating the wrapped modules (Generator instances) , each of those, representing a different module of our design

sort(...)
Method used for sorting the DFS' result

prepareResults(...)
Method used for setting the attributes of the dfs_results struct instance.
This method must be called after calling the sort on the DFS' results.

preparePlaintext(...)
Method used for setting the plaintext string field of the dfs_results.

prepareXML(...)
Method used for setting the XML string field of the dfs_results.

prepareJSON(...)
Method used for setting the JSON string field of the dfs_results.

prepareCSV(...)
Method used for setting the CSV string field of the dfs_results.

10.2.3 Public Methods

```
public:
//Constructors
DesignAnalyzer(RTLIL::Design *design);
//Printers
void PRINT_allModules();
void PRINT_allModulesSupportGraphs();
void PRINT_CoupledSignals();
 Final_Results   DFS(bool enableDeepScan, bool condense=false, bool pt=false, bool xm=false, bool js=false, bool cs=false);
```

10.2.4 Constructor

```
DesignAnalyzer(RTLIL::Design *design);
```

DesignAnalyzer(...)
This constructor takes in input the RTLIL::Design because it uses it for building the list of wrapped modules (Generators related to design's modules).

10.2.5 Printers

These methods are used for printing data related to converted modules.

```
void PRINT_allModules();
void PRINT_allModulesSupportGraphs();
void PRINT_CoupledSignals();
 Final_Results   DFS(bool enableDeepScan, bool condense=false, bool pt=false, bool xm=false, bool js=false, bool cs=false);
```

PRINT_allModules(...)
Method that prints all the infos related to (wrapped) modules of the design.

PRINT_allModulesSupportGraphs(...)
Method that prints all the internal structures (supportGraph) of the wrapped modules.

PRINT_CoupledSignals(...)
Method that prints all the coupled signals of each (wrapped) module of the design.

DFS(...)
Method that return the global variable filled with all the string attributes setted for printing the results on persistent formats.

Part III
Utilities

This part is about those files that contains functions that we call in several parts of our code and works tidly with standard Yosys RTLIL components.
They're used for expanding the knowledge about Cells parameters in order to define how they can alter (or not) a signal and for dumping certain parts of the RTLIL::Design for general purposes.

```
        //files
        CellRepository.h
        rtlil_dumper.h
```

11 CellRepository

This namespace contains all the data structures used for deciding the type and characteristics of a standard (non-module) cells.

Accessibility:

File: CellRepository.h
Namespace: CellRepository

Contents:

```
//Enums
enum CellCategory{...};
//Structs
struct CellStructure{...};
struct CellRepository{...};
```

11.1 CellCategory enum

The purpose of this enum is to define constants used to decree which types of cell are we analyzing , based on the operations that it performs.

External usage: CellCategory::CellCategory

```
//The cell purpose is to perform some kind of logical operation on data inputs
LOGICAL_OPERATOR = 0,
//The cell purpose is to perform some kind of arithmetical operation on data inputs
ARITHMETIC_OPERATOR = 1,
//The cell purpose is to perform operations on bit of data inputs
BITWISE_OPERATOR = 2,
//The cell purpose is to perform comparison on data inputs
COMPARISON_OPERATOR = 3,
//The cell purpose is to select among a set of possible data inputs
MULTIPLEXER_OPERATOR = 4,
//The cell purpose is to retain a value
RETENTION_OPERATOR = 5,
//The cell purpose is to reduce a vector to a scalar
REDUCTION_OPERATOR = 6,
//The cell purpose is to shift all the bits of inputs up to a defined value
SHIFT_OPERATOR = 7,
//The operator perform a specific operation (which is not part of any other cathegory i.e. alu which
      can be both logical, mathematical, ...) on data inputs
DEDICATED_OPERATOR = 9,
```

11.2 CellStructure struct

This struct is used for setting and all the standard cells provided by Yosys based on their type and ports.

External usage: CellCategory::CellStructure

11.2.1 Private Attributes

```
    private:
        //Type of the cell
        std :: string  type  ;
        //Which operation the cell can perform
        CellCategory  category  ;
        //Boolean flag used for decide is a cell is a cell that its purpose is to change the signal in
            any way
        bool signal_changer  ;
        //List of data input ports for a cell
        std :: vector <std:: string > data_inputs;
        //List of selection input ports for a cell
        std :: vector <std:: string > selection_inputs ;
        //List of output ports of a cell.
        std :: vector <std:: string > outputs;
```

11.2.2 Private Methods

Since all these methods are simple setters , used for setting the attributes listed above , we think that they're usage is enought self explanatory.

```
private:
    void setType(std :: string  type );
    void setCategory(CellCategory  category );
    void setSignalChanger(bool signal_changer);
    void setDataInputs( std :: vector <std:: string > data_inputs);
    void  setSelectionInputs ( std :: vector <std:: string > selection_inputs );
    void setOutputs(std :: vector <std:: string > outputs);
```

11.2.3 Public Methods

```
    public:
    //Constructors
     CellStructure ( std :: string  type, CellCategory  category, bool signal_changer, std :: vector <std:: string > data_inputs, std :: vector <std:: string
         > selection_inputs , std :: vector <std:: string > outputs);
    //Getters
    std :: string  getType ( );
    CellCategory  getCategory ( );
    bool getSignalChanger();
    std :: vector <std:: string > getDataInputs( );
    std :: vector <std:: string > getSelectionInputs ( );
    std :: vector <std:: string > getOutputs();
    //Utilities
    bool isDataInput(std :: string  unknown_port);
    bool isSelectionInput ( std :: string  unknown_port);
    bool isOutput(std :: string  unknown_port);
    bool isLogicalOperator ( );
    bool isArithmeticOperator ( );
    bool isBitwiselOperator ( );
    bool isComparisonOperator();
    bool isMultiplexerOperator ( );
    bool isRetentionOperator ( );
    bool isReductionOperator( );
    bool isShiftOperator ( );
    bool isDedicatedOperator( );
    //toString()
    std :: string  toString ( );
```

CellStructure(...)
This constructor is used for creating a wrapper for Yosys standard cells.

The getter methods are related to the struct attributes listed above.

The utilities methods , instead , returns a boolean value that defines if a specific cell is used as the type of operator specified by the method name itself.

11.3 CellRepository struct

This struct is used for analyzing the cells that are passed as nodes from outer parts of the projects in order to decide which the nature of the node or the ports associated.

11.3.1 Private Attributes

In the section below , only the relevants attributes has been added and explained.

```
private:
        //List that contains all the CellStructures that are associated to YOSYS cells
        std :: vector <CellStructure *> cells_repository  ;
```

11.3.2 Private Methods

```
private:
        void setUnaryOperatorCellStructure(std :: string  name,CellCategory  category , bool signal_changer);
        void setBinaryOperatorCellStructure (std :: string  name, CellCategory  category , bool signal_changer);
        void setMuxCellStructure(std :: string  name);
        void buildCellRepository ();
```

setUnaryOperatorCellStructure(...)
Method used for setting unary yosys cells into the new format.

setBinaryOperatorCellStructure(...)
Method used for setting binary yosys cells into the new format.

setMuxCellStructure(...)
Method used for setting mux yosys cells into the new format.

buildCellRepository(...)
Method used for wrapping all the Yosys RTLIL::Cells into our data format.

11.3.3 Public Methods

```
public:
        //Constructors
        CellRepository () ;
        //Getters
        std :: vector <CellStructure *> getCellRepository();
         CellStructure  *getCell( std :: string  cellType )
```

getCellRepository(...)
Method that returns the entire cell repository (all the wrapped RTLIL::Cell associated to atomic components).

getCell(...)
Method used for getting a specific (wrapped) cell based on its type.

12 RTLIL_DUMPER

Namespace that contains some utilities functions for translating certain components of the Yosys RTLIL in our internal representation corresponding objects.

Accessibility:

File:	rtlil_dumper.h
Namespace:	RTLIL_DUMPER

This file doesn't contains any data structure, only functions.

12.0.1 Functions

```
public:
        bool isValidPortID(std :: string  portID);
        std :: pair <bool, int> portIDDumper(std::string portID);
        std :: pair <bool,Signal::Signal> sigChunkDumper(const RTLIL::SigChunk &chunk);
        std :: pair <bool, std::vector<Signal::Signal> > sigSpecDumper(const RTLIL::SigSpec &sig);
        std :: pair <std::vector<Signal::Signal>,std:: vector <Signal::Signal> > connDumper(const RTLIL::SigSpec &left,const RTLIL::SigSpec &
                right);
```

isValidPortID(...)
Method used for checking if an unnamed port associated to module has associated a valid id (return true) or not(return false).

portIDDumper(...)
Method used for converting a string (Which represent an RTLIL::IdString of a port) into an integer.

sigChunkDumper(...)
Method used for converting an RTLIL::SigChunk associated to a signal to a Signal.
The first value of the pair (bool) is used to state if it was possible to convert the Chunk to a Signal.
result.first = true *implies* result.second = Signal
result.first = false *implies* result.second = void signal -¿ no solution found

sigSpecDumper(...)
Method used for converting an RTLIL::SigSpec into a vector of Signals (i.e. the chunks associated to the SigSpec).
The first boolean flag of the pair states if we have reached a solution or (as in case of all constants) not.

connDumper(...)
Method used for converting an a connection among two RTLIL::Cell or modules into a pair of vectors that represents the chunks associated to such connections.
Since the connections are based on (paired) wires , we will always have that the two vectors are equals since all the signals are related.

Part IV
Standard C++11 Support Utilities

This part contains some utilities created for simplify the handling of standard C++11 constructs like strings and vectors.
They're not directly related to Yosys but we use them on several part of our module for simplify some tasks.

```
        //files
        stdio_support.h
        stdstring_support.h
        stdvector_support.h
```

13 IOSupport

Namespace that contains a set of functions used for simplify the usage of the system's I/O .
We use the methods provided by this namespace for perfoming the printing of results on files.

This namespace doesn't contains any data structure , only functions.

Accessibility:

File:	stdio_support.h
Namespace:	IOSupport

13.0.1 Functions

```
public:
      //Opens FILENAME and write results in there.
      void writeResultToCleanFile( std :: string  filepath , std :: string  results );
      //Reset the content of a file.
      void resetFile ( std :: string  filepath );
      //Append results to FILENAME.
      void appendToFile(std :: string  filepath , std :: string  results );
```

14 StringSupport

Namespace that contains a set of functions used for simplify the usage of strings inside our module.

This namespace doesn't contains any data structure , only functions and constants.

Accessibility:

File:	stdio_support.h
Namespace:	IOSupport

14.0.1 Functions

```
public:
        void findAndReplaceAll(std:: string  &data, std :: string  toSearch, std :: string  replaceStr );
        std :: string  cleanse( std :: string  dirty , bool keepWhitespaces=false);
        std :: vector <std:: string > cleanse(std :: vector <std:: string > dirty);
        std :: string  substr_delimiter ( std :: string  toFilter , std :: string  startDelimiter  = DIMENSION_START, std::string endDelimiter =
                DIMENSION_END);
        bool startsWith( std :: string  haystack, std :: string  needle );
        std :: vector <std:: string > split( std :: string  input, char delimiter = WHITESPACE_CHAR, bool deleteEmpty=false, bool keepWhitespaces
                =false);
        std :: vector <std:: string > split( std :: string  input, std :: string  delimiter , bool removeFirst=false, bool cleanse=false);
        std :: pair <int, int > extractPair(std::string  pair , bool verbose=false);
        std :: pair <int, int > singleSignalExtraction(std:: string  signal );
        std :: vector < std:: pair < std:: string , std :: pair <int, int > > > extractSignals(std::string toSplit, bool verbose=false);
        bool isSingleBit( std :: pair < std :: string , std :: pair <int, int > > signal );
        bool isSingleBit( std :: string  signal  );
        std :: string  DUMP_vector(std:: vector <std:: string > vec, bool dividePrint=false);
        std :: string  DUMP_vector(std:: vector < std:: pair <std:: string , std :: string > > vec, bool dividePrint=false);
```

findAndReplaceAll(...)
Method used for replacing all the occurences of toSearch with replaceStr

cleanse(...)
Method used to cleanse the string for signal renaming analysis.

cleanse(std::vector...)
Method used to cleanse the whole vector content.

substr_delimiter(...)
Returns the substring of toFilter between startDelimiter and endDelimiter

startsWith(...)
Method used to decree if a certain string starts with a certain character

split(...)
Method used for splitting a string with a certain character delimiter

extractPair(...)
From the string of the shape [x,y] return x and y within the std::pair.

singleSignalExtraction(...)
Extract all the info about a signal.

extractSignals(...)
Given a bunch of signal informations, return them in an organized way, highlithing both name and used bit of that signal

isSingleBit(std::pair...)
Is a single bit signal? ¡for pair¿

isSingleBit(std::string...)
Is a single bit signal? ¡for string¿

DUMP_vector(...)
Returns a string representation of the string vector.

15 VectorSupport

Namespace that contains a set of functions used for simplify the usage of std::vectors inside our module.

This namespace doesn't contains any data structure , only functions.

Accessibility:

File:	stdio_vector.h
Namespace:	VectorSupport

15.0.1 Functions

```
public:
        std :: vector <T> append(std::vector<T> source, std::vector<T> toAppend);
        bool isIn(std :: vector <T> haystack, T needle);
        std :: vector <T> conditionalInsert(std :: vector <T> vector, T toInsert);
        std :: vector <T> merge(std::vector<T> left, std::vector<T> right);
        std :: vector <T> minus(std::vector<T> inHere, std::vector<T> notInHere);
        bool equals(std :: vector <T> first, std :: vector <T> second);
        void pop_n( std :: vector <T>& vec, int n=0);
```

append(...)
Function that safely concatenate two vectors.

isIn(...)
Function that performs a linear search in a vector

conditionalInsert(...)
Function that performs an insert with no repetitions.

merge(...)
Function that merge the given vectors with no repetitions.

minus(...)
Function that returns the vector of elements which are in the first but not in the second vector.

equals(...)
Function that returns the vector of elements which are in the first but not in the second vector.

pop_n(...)
Function that delete n-th element

Part V
External Yosys Modules

In this section , we have decided to briefly explains which are the external Yosys files that we have slightly modified in order to make our application.

```
//files
circuit.h
circuit.cc
module_to_graph.h
```

This files comes from the default Yosys GIT repository and we have only slightly modified them in order to allow our application to extract data from the Yosys RTLIL::Design.

16 SubCircuit_v2

This namespace contains an implementation of the Ullmann Subgraph Isomorphism algorithm for coarse grain logic network.

We have used its provided data structure in order to keep track of the internal structure of a node and , also , for managing the connections between nodes in our design.

Accessibility:

File:	circuit.h & circuit.cc
Namespace:	SubCircuit_v2

This namespace only contains at a top level a class (Graph), used for representing the data structure parts of the RTLIL::Design corresponding to an RTLIL::Module .

16.0.1 Graph Class brief explanations

```
protected:
        //Data structure used for keeping tracks of bit-level infos
        struct BitRef{...};
        //Data structure used for keeping tracks of connections between nodes
        struct Edge{...};
        //Data structure used for keeping tracks of bit infos related to a node port
        struct PortBit{...};
        //Data structure used for keeping tracks of infos regarding ports
        struct Port{...}
        //data structure used for keeping tracks of infos regarding nodes
        struct Node{...}
        //Boolean flag used to decree if the current ports are all connected to module's level signals
        bool allExtern;
        std::map<std::string, int> nodeMap;
        //List of nodes of the current module
        std::vector<Node> nodes;
        //List of edges of the current module
        std::vector<Edge> edges;
public:
        //Constructors
        Graph();
        Graph(const Graph &other, const std::vector<std::string> &otherNodes);

        //Method used for creating a node filling the corresponding data structure described above
        void createNode(std::string nodeId, std::string typeId, void *userData=NULL, bool shared=false);
        //Method used for creating a port filling the corresponding data structure described above
        void createPort(std::string nodeId, std::string portId, int width=1, int minWidth=-1);
        //Method used for creating a connection based on edges
        void createConnection(std::string fromNodeId, std::string fromPortId, int fromBit, std::string toNodeId, std::string toPortId, int toBit
                , int width=1);
        void createConnection(std::string fromNodeId, std::string fromPortId, std::string toNodeId, std::string toPortId);
        //Method used for adding constant values ro a port (which means that such port will not be binded to a
                specific signal)
        void createConstant(std::string toNodeId, std::string toPortId, int toBit, int constValue);
        void createConstant(std::string toNodeId, std::string toPortId, int constValue);
        //Method used to state if a certain port is connected with module's level signals
        void markExtern(std::string nodeId, std::string portId, int bit=-1);
        //Method used for state that all the ports are connected to module's level signals
        void markAllExtern();
        void print();
```

For a more detailed explanation please check the original's yosys files (subcircuit.cc, subcircuit.h) on the official Yosys repository.

17 ModToGraph

This namespace contains a single function : the module_to_graph function , which we have used for extracting info about an RTLIL::Module in order to fill our data structure with the corresponding data of interest.

Also this function is part of the subcircuit.cc/subcircuit.h files of the original standard Yosys repository.

Accessibility:

File:	module_to_graph.h
Namespace:	ModToGraph

This namespace contains only the definition of a support structure for working with bits :

```
struct bit_ref_t {...};
```

and the function that we have explained above

```
//Function used for extracting data from an RTLIL::Module and fill the SubCircuit_v2 passed as
       parameter
std :: vector < std:: pair < std:: string , int > > mod2graph(SubCircuit_v2::Graph &graph, RTLIL::Module *mod);
```