CSLOM Elab

1. Eval
2. Expression Link
3. Expression
4. Methods / commands
5. Autorouter
6. Adress checking / generation, doc. gen, C++ & Verilog code gen.

Note: We need a set of casting operators

Supported numbers in CSLC: INT 32, INT Big, Fixed Floating Point, Fixed Floating Point Big, Floating Point, Floating Point Big, Verinum

---

a = 4
b = 3
c = 10
d = 11
Cmd Shell

Debug Display (patent this):
set_width(a+b+c-d);
inputs->4+3+10-11
result->6

Eval must be checked for carry and overflow
Fixed point -> truncation
-> rounding

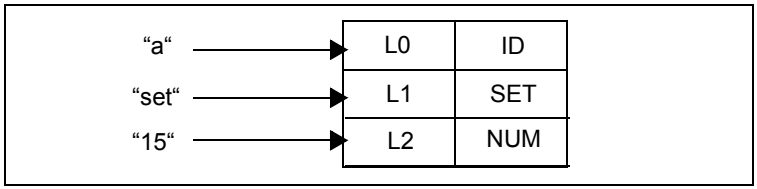
We need 3 different number classes: 32 bit numbers, IEEE float 80 bit and fixed point numbers. Or we can get a library that handles big numbers (must have correct license: no GPL, no LGPL)

method to be added to CSLOM tree: bool noXZ(string);
and: int convertStringToNumber(string);

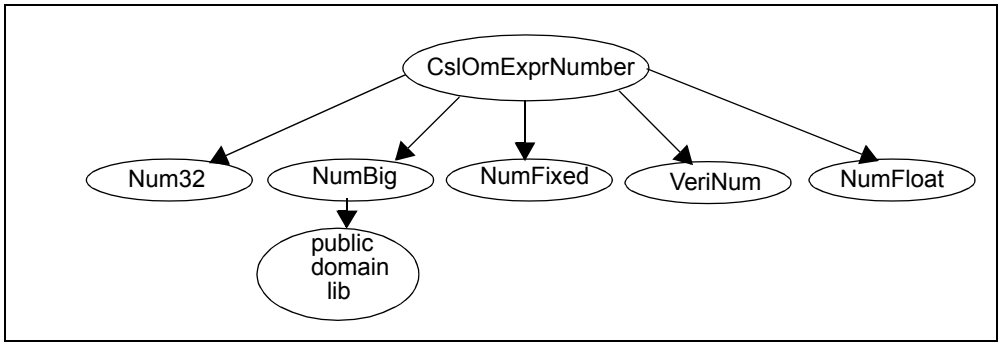
We need to check numeric range supported by commercial compilers: VXL, Verilog

---

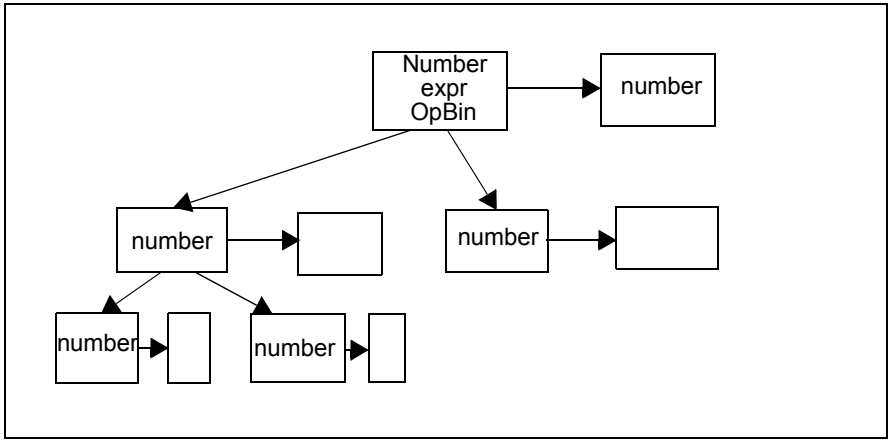
Only create a Verinum if the number string has X or Z
The Tree Walker or CSLOM sets the hasXZ() bit for the current OM expr node if the current node has one of the following conditions:
1. The node is a number and it has an X or Z digit
2. An expr operator has a child with the has XZ boolean set.



If any leaf node in the expr tree has hasXZ then set all nodes in expr tree to hasXZ.



class -> command
execute
check

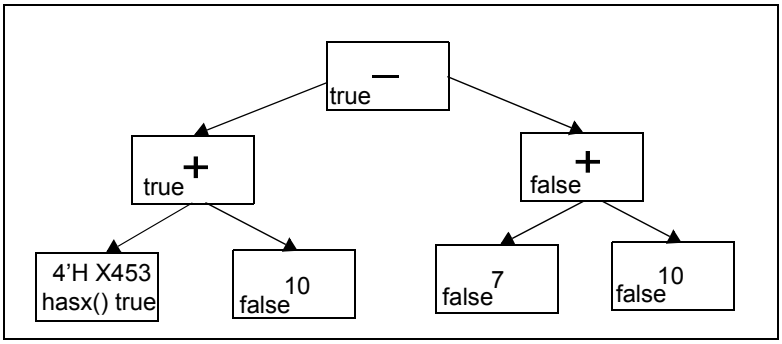


---
preprocessor
parse
tree-walker
Build The OM
---
eval const expr
check
Note: OM has constants and comments

tree picture

at each expression if is const then create a Number class at each node

---
We need to add for numerical eval the hasXZ=false parameter



calling Eval Visitor from the root of the constant expression:

```
if(node->hasXZ()) {
  VisitorVerinumEval(node);
}
else
{
  VisitorNumericEval(node);
}
```

We need a list of all public domain packages in a datasheet: like XERCES, ANLTR, BIGNumber etc:

Package Name | License Type | Requirements

We need to add a number type bit to the CSLOmExpr base class:

	hasXZ	const
INT32		T
BigNUM		T
Float		T
Fixed		T
Verinum	T	T

1.1 Verinum rules

1.

All numbers are stored in 2 formats string and bit array

2.

If the expression being computed is an assign statement:

- base of the result is the base of the LHS expression
- width of the result is the width of the LHS expression

3.

If the expression determines the width a setwidth(x+y+z) where x,y and z are umeric constants then the width is computed for each operator in the expression tree.

For example:

carryout equals <<width is the result of the shift op.
+ 0 add 1 bit
+ 1
* compute the result of the op that is the width
If carry out is set then hte result of the operation is one bit wider than the widest expression in the operator tree.

use IEEE 1384

truth table + - / * % << >> (0)..... = carry bit 0

0000	0	000
0001	0	101
0010	0	x
.....	0	z
0011	1	001
.....	1	111
0100	1	x
.....	1	z
	x	0
case statement	x	1
selector	x	x
	x	z
	z	0
	z	x
	z	z

one case statement for operator
fill in the vals

Standalone expressions non-constants

a * b width(maxValue(a)*maxValue(b))
a + b max(width(a),width(b))+1
a << b width(maxValue(a) << maxValue(b))

---

statements
non-constants
LHS=RHS
width(LHS)
of LHS&RHS
Formats determines whether the LHS or RHS bits are dropped

LHS = a op b
fixed fixed fixed

ExprBase LHS
1.declared width
2.computed width
3.carry out
4.hasXZ
5.truncated
6.rounded
7.enumExprSide{LHS,RHS}
8.enum castType{int 32,int BigNumber,Fixed32.....}
9.enum inferredType{int 32,int BN,Fixed 32.....}

numeric_const<Int 32>(Numeric expression)

type numeric expression result type is cast to type

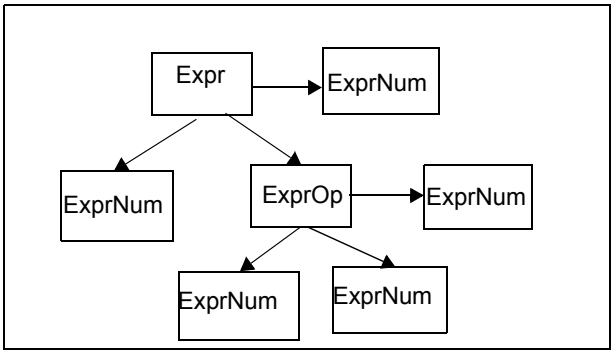
---
clamp(32,x)
if x<=32 then y=x;
else if x>32 then y=32

Visitor

if number getVal returns #
else if operator

1.call getVal an n children where n is the number of children
2.compute the result of operator (getVal(children 1.....n))
3compute the cast type and other attribute types of the result

4.create a ExprNum node using the result computed
5.set the ExprNum* equal to the new " "node



Truncation of the left hand bits in an expression result

1.LHS expression width is less than the RHS expression width.

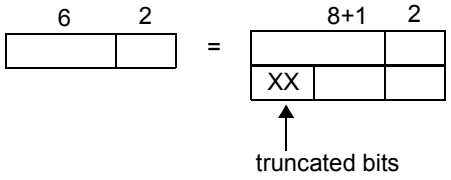
The LHS expression type is Int or fixed point or float

EXAMPLE :

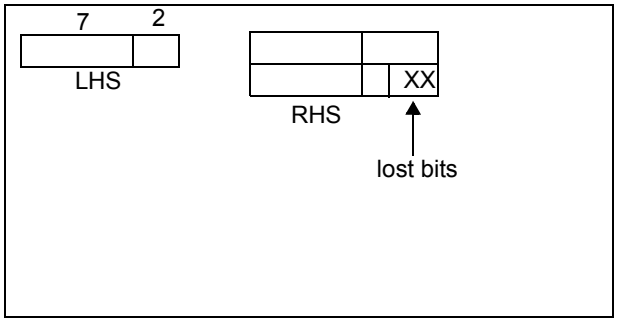
Width is shown in paranthesis

fixed(6.2,8) = fixed(8.2,10) + fixed(6.2,8)

However the upper 2 bits are lost since the LHS format is 8+1(add one bit according to the fixed point arithmetic rules)



2.Rounding the fractional bits



ExprBase
bool hasXZ
bool isConst
ExprNum*
bool isEvaluated
int width
getWidth
calcWidth
maxValue
isTrunk
int noOfBitsTrunk
isRounded
int noOfBitsRounded

---
type of Num
fractional size
mantisa size
decimal size
signed bit