Logic Gate Primitives in myHDL

Steven K Armour

Table of Contents

1 Libraries and Helper functions

In [1]:

```
#This notebook also uses the `(some) LaTeX environments for Jupyter`
#https://github.com/ProfFan/latex_envs wich is part of the
#jupyter_contrib_nbextensions package

from myhdl import *
from myhdlpeek import Peeker
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

from sympy import *
init_printing()

import itertools

#EE drawing tools in python from https://cdelker.bitbucket.io/SchemDraw
import SchemDraw as schem
import SchemDraw.elements as e
import SchemDraw.logic as l

#https://github.com/jrjohansson/version_information
%load_ext version_information
%version_information myhdl, myhdlpeek, numpy, pandas, matplotlib, sympy
```

Out[1]:

Software	Version
Python	3.6.2 64bit [GCC 4.4.7 20120313 (Red Hat 4.4.7-1)]
IPython	6.2.1
OS	Linux 4.15.0 30 generic x86_64 with debian stretch sid
myhdl	0.10
myhdlpeek	0.0.6
numpy	1.13.3
pandas	0.23.3
matplotlib	2.1.0
sympy	1.1.2.dev
itertools	The 'itertools' distribution was not found and is required by the application
SchemDraw	0.3.0
	Mon Sep 03 14:42:23 2018 MDT

In [2]:
```
#helper  functions to read in the .v and .vhd generated files into pyth
def VerilogTextReader(loc, printresult=True):
    with open(f'{loc}.v', 'r') as vText:
        VerilogText=vText.read()
    if printresult:
        print(f'***Verilog modual from {loc}.v***\n\n', VerilogText)
    return VerilogText

def VHDLTextReader(loc, printresult=True):
    with open(f'{loc}.vhd', 'r') as vText:
        VerilogText=vText.read()
    if printresult:
        print(f'***VHDL modual from {loc}.vhd***\n\n', VerilogText)
    return VerilogText

def ConstraintXDCTextReader(loc, printresult=True):
    with open(f'{loc}.xdc', 'r') as xdcText:
        ConstraintText=xdcText.read()
    if printresult:
        print(f'***Constraint file from {loc}.xdc***\n\n', ConstraintTe
    return ConstraintText
```

In [3]:
```
def TruthTabelGenrator(BoolSymFunc):
    """
    Function to generate a truth table from a sympy boolian expression
    BoolSymFunc: sympy boolian expression
    return TT: a Truth table stored in a pandas dataframe
    """
    colsL=sorted([i for i in list(BoolSymFunc.rhs.atoms())], key=lambda
    colsR=sorted([i for i in list(BoolSymFunc.lhs.atoms())], key=lambda
    bitwidth=len(colsL)
    cols=colsL+colsR; cols

    TT=pd.DataFrame(columns=cols, index=range(2**bitwidth))

    for i in range(2**bitwidth):
        inputs=[int(j) for j in list(np.binary_repr(i, bitwidth))]
        outputs=BoolSymFunc.rhs.subs({j:v for j, v in zip(colsL, inputs
        inputs.append(int(bool(outputs)))
        TT.iloc[i]=inputs

    return TT
```

2 Table of Digital Gate Symbols commonly used

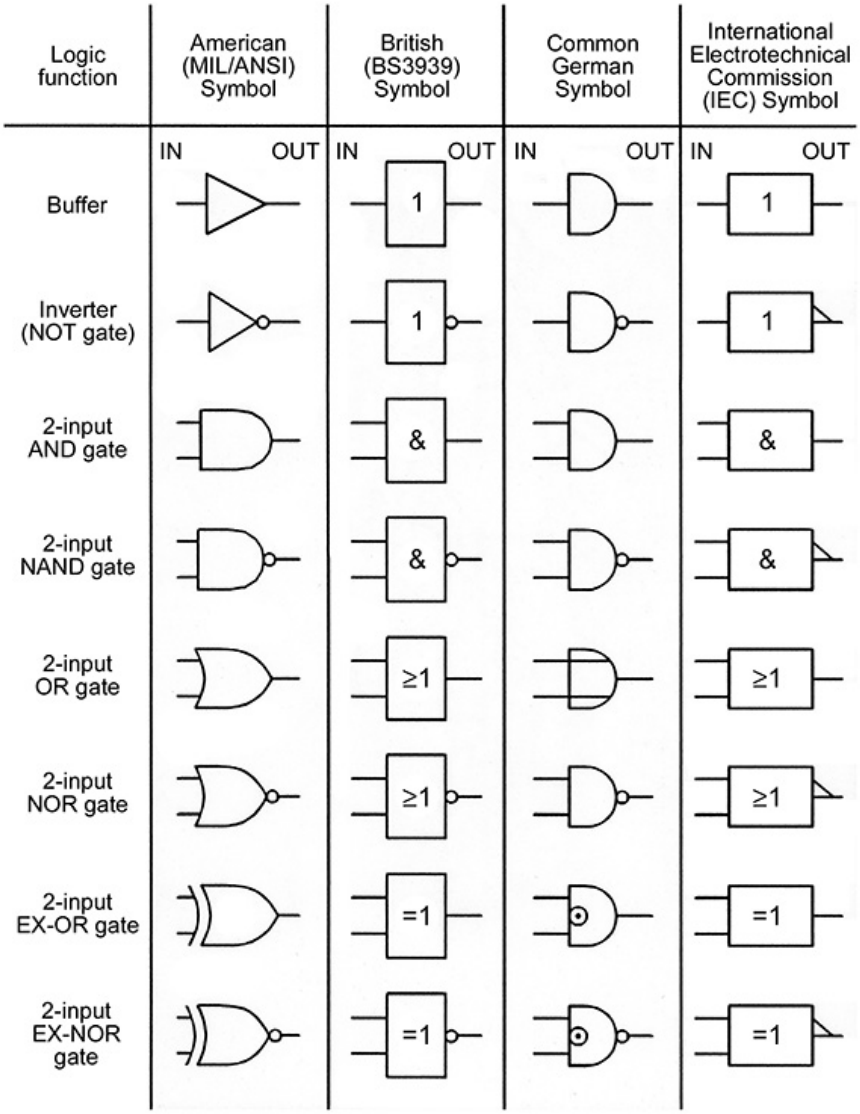


Figure 1: Table of selected Logic Gate Symbols found worldwide;

3 How to load output Verilog from myHDL to Xilinx's Vivado

YouTube (https://youtu.be/K6rLsdJO2CE)

4 Wire High

Definition 1 *\label{def:Wire High} A Wire to High is a Wire connected to the Highest reference voltage in a Digital Circuit with the Boolean equivalency of*

$$Y = 1$$

that of ***always True***

In [4]:
```
d = schem.Drawing(unit=.5)
d.add(e.VDD, label='High')
d.add(e.LINE, l=d.unit*2, d='down')
d.add(e.DOT_OPEN, botlabel='$1$')
d.draw()
```

4.1 Sympy Exspresion

In [5]:
```
Y=symbols('Y')
YEq=Eq(Y, 1); YEq
```

Out[5]: $Y = 1$

4.2 myHDL Module

In [6]:
```
#decorator to specify a myHDL module that can be converted
#to myHDL
@block
def WireHigh(Y):
    """
    Wire to Constatnt High
    Input:
        None
    Output:
        Y(bool): output
    """
    #create signal to High Source
    HSource=Signal(bool(1))

    #decorator for a compintorical senstvity
    @always_comb
    def logic():
        #internal function to specfiy Out is set to High Source
        Y.next=HSource

    #return all internal functions
    return instances()
```

4.3 Verilog Conversion

In [7]:
```
#create output signal to bind to `WireHigh` Mod
Y=Signal(bool(0))
#Create instatnce of `WireHigh` and bind `Out` Signal to it
DUT=WireHigh(Y)
#Run convert on `DUT` instance of `WireHigh` to create Verilog/VHDL
DUT.convert()
#Read ouput `WireHigh.v` into notebook
VerilogTextReader('WireHigh');
```

Verilog modual from WireHigh.v

```
 // File: WireHigh.v
// Generated by MyHDL 0.10
// Date: Mon Sep  3 14:42:38 2018


`timescale 1ns/10ps

module WireHigh (
    Y
);
// Wire to Constatnt High
// Input:
//     None
// Output:
//     Y(bool): output

output Y;
wire Y;

wire HSource;

assign HSource = 1'd1;



assign Y = HSource;

endmodule
```

```
/home/iridium/anaconda3/lib/python3.6/site-packages/myhdl/conversion/_t
oVerilog.py:349: ToVerilogWarning: Signal is not driven: HSource
  category=ToVerilogWarning
```

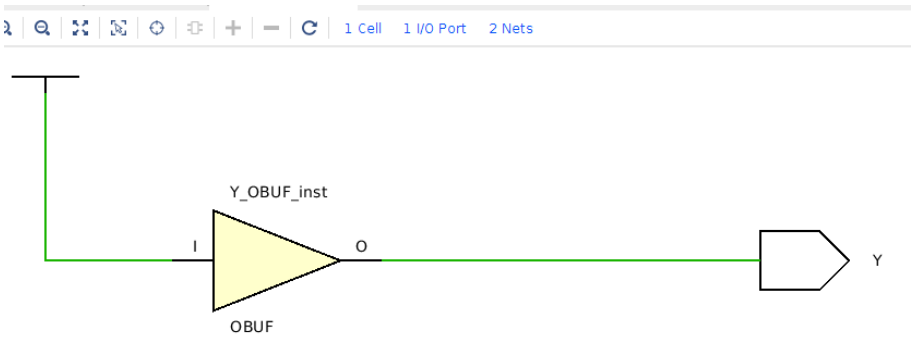


Figure 2: WireHigh RTL schematic (Ignore the Buffer in the middle); Xilinx Vivado 2017.4

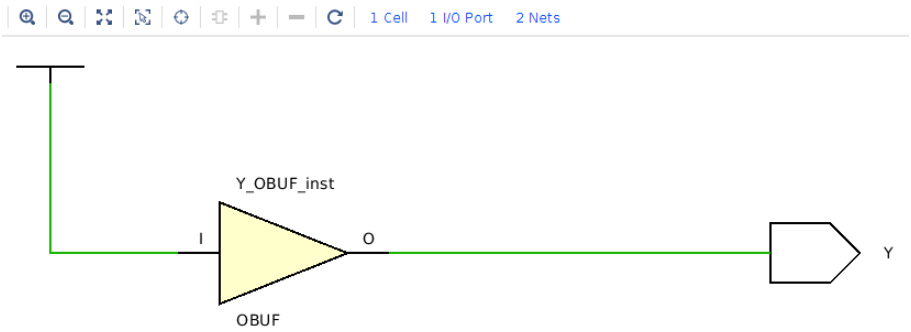


Figure 3: WireHigh Synthesized Schematic (Ignore the Buffer in the middle); Xilinx Vivado 2017.4

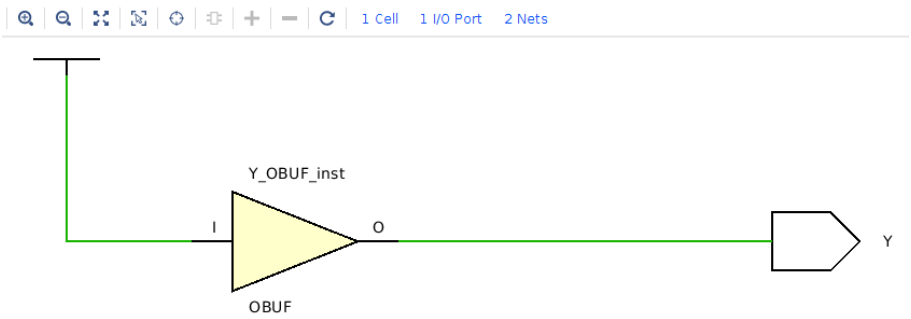
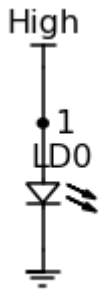


Figure 4: WireHigh Implementated Schematic (Ignore the Buffer in the middle); Xilinx Vivado 2017.4

4.4 PYNQ-Z1 Deployment

4.4.1 Board Circuit

In [8]:
```
d = schem.Drawing(unit=.5)
d.add(e.VDD, label='High')
d.add(e.LINE, l=d.unit*2, d='down')
d.add(e.DOT, rgtlabel='$1$')
d.add(e.LINE, d='down', l=d.unit*2)
d.add(e.LED, rgtlabel='LD0', d='down')

#LED to ground
d.add(e.LINE, d='down', l=d.unit*1.5)
d.add(e.GND)
d.draw()
```



4.4.2 Board Constraint

In [9]:
```
ConstraintXDCTextReader('PYNQ_Z1Constraints_WireHigh');
```

```
***Constraint file from PYNQ_Z1Constraints_WireHigh.xdc***

 ## PYNQ-Z1 Board Constraints for WireHigh.v
## Based on https://github.com/Digilent/digilent-xdc/blob/master/Arty-Master.xdc (https://github.com/Digilent/digilent-xdc/blob/master/Arty-Master.xdc)

##LED 0
set_property -dict {PACKAGE_PIN R14 IOSTANDARD LVCMOS33} [get_ports
 {Y}]
```

LED 0 (the furthest Right one above the buttons) will light up green

5 Wire Low

Definition 2 *A Wire to Low is a Wire connected to the Lowest reference voltage in a Digital Circuit with the Boolean equivalency of*

$$Y = 0$$

that of ***alway False***

In most modern digital circuits this Lowest reference voltage will be digital ground.

In [10]:
```
d = schem.Drawing(unit=.5)
d.add(e.DOT_OPEN, label='$0$')
d.add(e.LINE, l=d.unit*2, d='down')
d.add(e.GND)
d.draw()
```

5.1 Sympy Expression

In [11]:
```
Y=Symbol('Y')
YEq=Eq(Y, 0); YEq
```

Out[11]: $Y = 0$

5.2 myHDL Module

In [12]:
```
#decorator to specify a myHDL module that can be converted
#to myHDL
@block
def WireLow(Y):
    """
    Wire to Constatnt Low
    Input:
        None
    Output:
        Y(bool): output
    """
    #create signal to Low Source
    LSource=Signal(bool(0))

    #decorator for a compintorical senstvity
    @always_comb
    def logic():
        #internal function to specfiy Out is set to Low Source
        Y.next=LSource

    #return all internal functions
    return instances()
```

5.3 Verilog Conversion

In [13]:
```
#create output signal to bind to `WireHigh` Mod
Y=Signal(bool(0))
#Create instatnce of `WireHigh` and bind `Out` Signal to it
DUT=WireLow(Y)
#Run convert on `DUT` instance of `WireHigh` to create Verilog/VHDL
DUT.convert()
#Read ouput `WireHigh.v` into notebook
VerilogTextReader('WireLow');
```

Verilog modual from WireLow.v

```
 // File: WireLow.v
// Generated by MyHDL 0.10
// Date: Mon Sep  3 14:43:08 2018


`timescale 1ns/10ps

module WireLow (
    Y
);
// Wire to Constatnt Low
// Input:
//     None
// Output:
//     Y(bool): output

output Y;
wire Y;

wire LSource;

assign LSource = 1'd0;



assign Y = LSource;

endmodule


/home/iridium/anaconda3/lib/python3.6/site-packages/myhdl/conversion/_t
oVerilog.py:349: ToVerilogWarning: Signal is not driven: LSource
  category=ToVerilogWarning
```

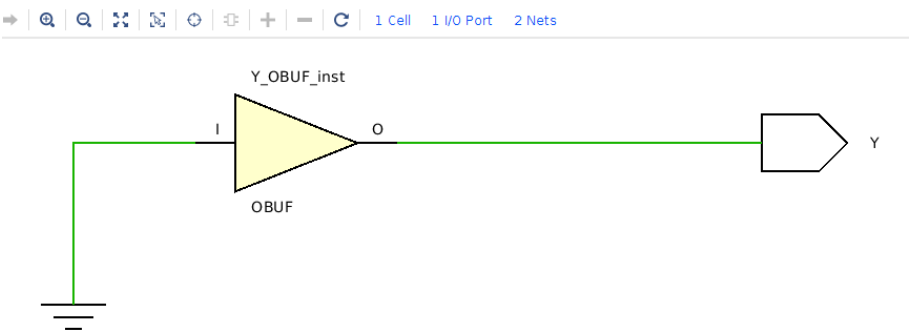


Figure 5: WireLow RTL schematic (Ignore the Buffer in the middle); Xilinx Vivado 2017.4

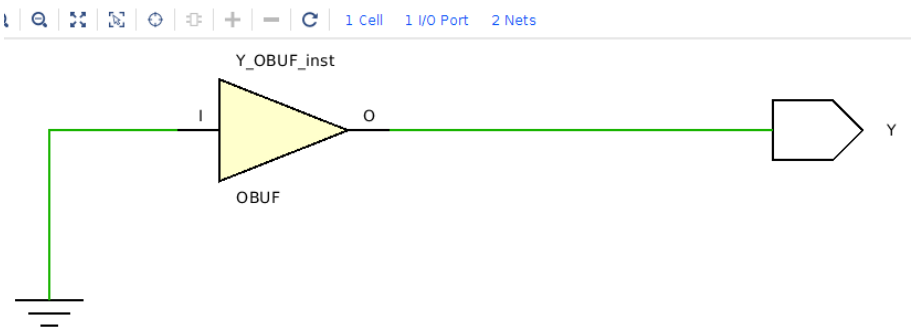


Figure 6: WireLow Synthesized Schematic (Ignore the Buffer in the middle); Xilinx Vivado 2017.4

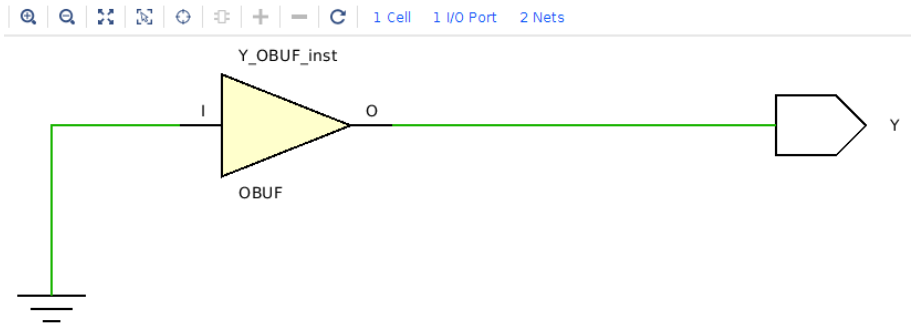
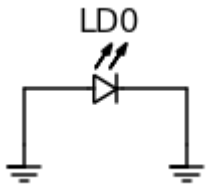


Figure 7: WireLow Implementated Schematic (Ignore the Buffer in the middle); Xilinx Vivado 2017.4

5.4 PYNQ-Z1 Deployment

5.4.1 Board Circuit

In [14]:
```
d = schem.Drawing(unit=.5)
LD0=d.add(e.LED, d='right', label='LD0')
d.add(e.LINE, l=d.unit*2, d='left', xy=LD0.start)
d.add(e.LINE, l=d.unit*2, d='down')
d.add(e.GND)

d.add(e.LINE, l=d.unit*2, d='right', xy=LD0.end)
d.add(e.LINE, l=d.unit*2, d='down')
d.add(e.GND)
d.draw()
```



5.4.2 Board Constraint

In [15]:
```
ConstraintXDCTextReader('PYNQ_Z1Constraints_WireLow');
```

```
***Constraint file from PYNQ_Z1Constraints_WireLow.xdc***

 ## PYNQ-Z1 Board Constraints for WireHigh.v
## Based on https://github.com/Digilent/digilent-xdc/blob/master/Arty-Master.xdc (https://github.com/Digilent/digilent-xdc/blob/master/Arty-Master.xdc)

##LED 0
set_property -dict {PACKAGE_PIN R14 IOSTANDARD LVCMOS33} [get_ports
 {Y}]
```

Nothing will happen since the Low Signal on the PYNQ-Z1 is ground so LED 0 will remain off

6 Buffer

Definition 3 *A Buffer Gate is a gate whose output is one to one with its input, in both behavior and ideally in time. The Buffer Gates boolean equivalence is*

$$Y = X$$

From a Boolean algebra/software perspective this may be a do-nothing gate but from a hardware perspective, it is incredibly valuable as it isolates the incoming signal circuit from any inline feedback on the outgoing signal's circuit. And is also typically used in circuit design as a way of ensuring the output of a system will be at the right voltage levels of the technology so that the next set of gates will register the input to them correctly

In [16]:
```
d = schem.Drawing(unit=.5)
d.add(e.DOT_OPEN, l=d.unit*3, lftlabel='$X$')
d.add(l.BUF, label='$Buffer$')
d.add(e.DOT_OPEN, l=d.unit*2, rgtlabel='$Y$')
d.draw()
```

6.1 Sympy Expression

In [17]:
```
X, Y=symbols('X, Y')
YEq=Eq(Y, X); YEq
```

Out[17]:

$$Y = X$$

In [18]:
```
TruthTabelGenrator(YEq)
```

Out[18]:

	X	Y
0	0	0
1	1	1

In [19]:
```
#convert the Sympy exspresion to a
#numpy exspresion via lambdify
YEqN=lambdify(X, YEq.rhs, dummify=False)
YEqN(np.array([0, 1]))
```

Out[19]:
```
array([0, 1])
```

6.2 myHDL Module

In [20]:
```
@block
def BufferGate(X, Y):
    """
    Buffer Gate demo module
    Input:
        X(bool): Input conection to wire between `X` and `Y`
    Ouput:
        Y(bool): Output connection to wire between 'X' and 'Y'
    """
    @always_comb
    def logic():
        Y.next=X
    return instances()
```

6.3 myHDL Testing

Every module (`WireHigh` and `WireLow` are just too primitive to test) should be tested with a Testbench in python. This test does not confirm that the module meets the requirements of synthesization in Verilog/VHDL but confirms that the behavior of the myHDL module under test (DUT) behavior wise conforms to the desired behavior.

Because of the data analysis ability of python to full analysis the test output in the same framework that the test, let alone module, was written in development time is decreased. This is one of the defining abilities of myHDL over other HDLs such as Verilog/VHDL

Synthesization in the world of HDL refers to the ability to turn HDL code into Hardware via an FPGA or a dedicated circuit (typically called an ASIC). It is for that reason that HDL codeing is not pure software nor pure hardware but rather a hybrid of the two. And why HDL coding has high constraints then all other coding seeing as the HDL must meet Behavior and Timing of software but must also meet the hardware synthesis requirements of Power, Physical Size, Implantability, and more

In [21]:
```
#generate random test values for BufferGate_TB
#stimules input In
TestLen=10
#for testing purpose the random genrator need to be seeded
#to create reproducable values
np.random.seed(12)
XTVs=np.random.randint(0,2, TestLen).astype(int)
XTVs
```

Out[21]:
```
array([1, 1, 0, 1, 1, 0, 1, 1, 0, 0])
```

In [22]:
```
#clear any previouse stored values in Peeker
Peeker.clear()
#Create the Signals to the BufferGate and
#add them to Peeker for display and anylis
X=Signal(bool(0)); Peeker(X, 'X')
Y=Signal(bool(0)); Peeker(Y, 'Y')

#create instance of `BufferGate` and bind signals
#to it as the "Device Under Testing"
DUT=BufferGate(X, Y)

#Create a Testbench for `BufferGate`
def BufferGate_TB():
    """
    myHDL only testbench for module `BufferGate`
    """
    #internal function that will apply
    #stimules to DUT
    @instance
    def stimules():
        for i in range(TestLen):
            #asign next stimules value to X input of DUT
            #for every 1 interval delay
            X.next=int(XTVs[i])
            yield delay(1)

        #flag to end the simulation of the DUT after all
        #stimules have been run
        raise StopSimulation()

    #return all internal stimules
    return instances()

#bind the DUT, Testbench and Peeker to simulator and run it
sim=Simulation(DUT, BufferGate_TB(), *Peeker.instances()).run()
```

In [23]:
```
#View Waveform diagram from Simulation with
#wavefroms in the order of `X` then `Y`
Peeker.to_wavedrom('X', 'Y')
```

In [24]:
```
#Capture the Waveform as a Pandas Dataframe for
#detiled anylsis
WireData=Peeker.to_dataframe()
#order the cols in the dataframe `X` then `Y`
WireData=WireData[['X', 'Y']]
WireData
```

Out[24]:

	X	Y
0	1	1
2	0	0
3	1	1
5	0	0
6	1	1
8	0	0

In [25]:
```
#apply the lampdfid sympy expspresion for a buffer to the X input to
#get a refrance to compare the BufferGate modules
WireData['YRef']=WireData['X'].apply(YEqN)
WireData
```

Out[25]:

	X	Y	YRef
0	1	1	1
2	0	0	0
3	1	1	1
5	0	0	0
6	1	1	1
8	0	0	0

In [26]:
```
#Test wither the modules ouput matchs the sympy refrance
Test=(WireData['Y']==WireData['YRef']).all()
print(f'Module `BufferGate` works as exspected: {Test}')
```

```
Module `BufferGate` works as exspected: True
```

6.4 Verilog Conversion

In [27]:
```
DUT.convert()
VerilogTextReader('BufferGate');
```

```
***Verilog modual from BufferGate.v***

 // File: BufferGate.v
// Generated by MyHDL 0.10
// Date: Mon Sep  3 14:43:49 2018


`timescale 1ns/10ps

module BufferGate (
    X,
    Y
);
// Buffer Gate demo module
// Input:
//     X(bool): Input conection to wire between `X` and `Y`
// Ouput:
//     Y(bool): Output connection to wire between 'X' and 'Y'

input X;
output Y;
wire Y;




assign Y = X;

endmodule
```

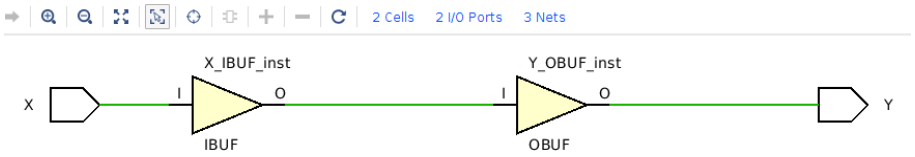


Figure 8: BufferGate RTL schematic; Xilinx Vivado 2017.4

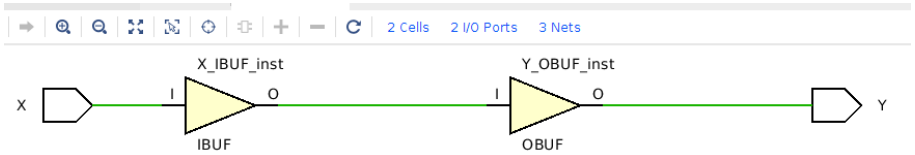


Figure 9: BufferGate Synthesized Schematic; Xilinx Vivado 2017.4

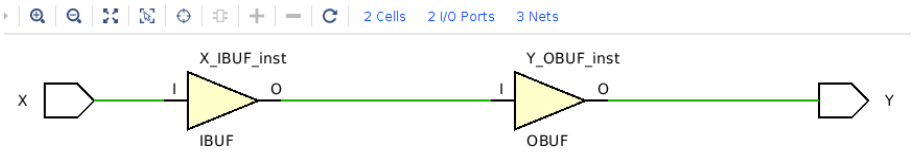


Figure 10: BufferGate Implementated Schematic; Xilinx Vivado 2017.4

6.5 myHDL to Verilog Testbench

In [28]:
```
#create BitVector for BufferGate_TBV
XTVs=intbv(int(''.join(XTVs.astype(str)), 2))[TestLen:]
XTVs, bin(XTVs)
```

Out[28]:
```
(intbv(876), '1101101100')
```

In [29]:

```
#Convertable testbenchs must be treated as a module
@block
def BufferGate_TBV():
    """
    myHDL -> Verilog testbench for module `BufferGate`
    """
    #create the signals to bind to the DUT
    X=Signal(bool(0))
    Y=Signal(bool(0))

    #print out the waveform of `X` and `Y`
    @always_comb
    def print_data():
        print(X, Y)

    #create a register to hold the testdata vector
    #Test Signal Bit Vector
    XTV=Signal(XTVs)

    #create an instance of the `BufferGate` and
    #bind X and Y to it
    DUT=BufferGate(X, Y)

    #create the stimules action
    @instance
    def stimules():
        for i in range(TestLen):
            #asign X the next bit from XTV testvector reg
            X.next=XTV[i]
            #delay one of the smallest time intervals
            yield delay(1)

        #raise the stop simulation flag
        raise StopSimulation()

    #return all internals of the Testbench
    return instances()

#create instance of the Testbench
TB=BufferGate_TBV()
#run the conversion to Verilog and insure that
#all Signal values are at there default value
TB.convert(hdl="Verilog", initial_values=True)
#read back the testbench into python
VerilogTextReader('BufferGate_TBV');
```

```
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
***Verilog modual from BufferGate_TBV.v***

 // File: BufferGate_TBV.v
// Generated by MyHDL 0.10
// Date: Mon Sep  3 14:43:59 2018


`timescale 1ns/10ps
```

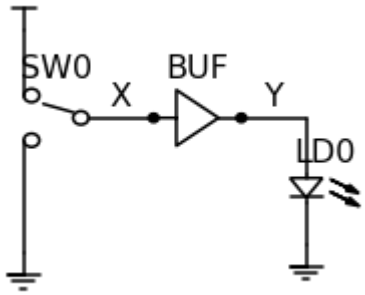
```
module BufferGate_TBV (

);
// myHDL -> Verilog testbench for module `BufferGate`


wire Y;
reg X = 0;
wire [9:0] XTV;

assign XTV = 10'd876;


always @(X, Y) begin: BUFFERGATE_TBV_PRINT_DATA
    $write("%h", X);
    $write(" ");
    $write("%h", Y);
    $write("\n");
end



assign Y = X;


initial begin: BUFFERGATE_TBV_STIMULES
    integer i;
    for (i=0; i<10; i=i+1) begin
        X <= XTV[i];
        # 1;
    end
    $finish;
end

endmodule


/home/iridium/anaconda3/lib/python3.6/site-packages/myhdl/conversion/_t
oVerilog.py:349: ToVerilogWarning: Signal is not driven: XTV
  category=ToVerilogWarning
```

6.6 PYNQ-Z1 Deployment

6.6.1 Board Circuit

In [30]:

```
d=schem.Drawing(unit=.5)
#add switch
SW0=d.add(e.SWITCH_SPDT2, reverse=True, label='SW0')

#connect to buffer
d.add(e.LINE, d='right', xy=SW0.a, l=d.unit*2, label='X')
d.add(e.DOT)
B0=d.add(l.BUF, label='BUF')

#buffer to LED
d.add(e.DOT)
d.add(e.LINE, d='right', l=d.unit*2, label='Y')
d.add(e.LINE, d='down', l=d.unit*2)
d.add(e.LED, rgtlabel='LD0', d='down')

#LED to ground
d.add(e.LINE, d='down', l=d.unit*1.5)
d.add(e.GND)

#swtich bottom to ground
d.add(e.LINE, d='down', xy=SW0.c, l=d.unit*4)
d.add(e.GND)


#switch top to rail
d.add(e.LINE, d='up', xy=SW0.b, l=d.unit*2.3)
d.add(e.VDD)
d.draw()
```



6.6.2 Board Constraint File

In [31]:
```
ConstraintXDCTextReader('PYNQ_Z1Constraints_BufferGate');
```

```
***Constraint file from PYNQ_Z1Constraints_BufferGate.xdc***

 ## PYNQ-Z1 Board Constraints for BufferGate.v
## Based on https://github.com/Digilent/digilent-xdc/blob/master/Arty-Master.xdc (https://github.com/Digilent/digilent-xdc/blob/master/Arty-Master.xdc)


## Switch 0
set_property -dict {PACKAGE_PIN M20 IOSTANDARD LVCMOS33} [get_ports
 {X}]


##LED 0
set_property -dict {PACKAGE_PIN R14 IOSTANDARD LVCMOS33} [get_ports
 {Y}]
```

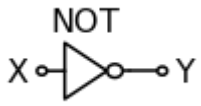
6.6.3 Video of Deployment

BUFFER Gate myHDL PYNQ-Z1 (YouTube (https://www.youtube.com/watch?v=v5q5OC0SwB4))

7 NOT

Definition 4 *A NOT Gate, also called an Inverter, is a logic gate that produces the negated value of the input. Its boolean expression is typically $$Y=\overline{X}$$ using the overhead bar negation notation.*

In [32]:
```
d = schem.Drawing(unit=.5)
d.add(e.DOT_OPEN, l=d.unit*3, lftlabel='$X$')
d.add(l.NOT, label='$NOT$')
d.add(e.LINE)
d.add(e.DOT_OPEN, rgtlabel='$Y$')
d.draw()
```



7.1 Sympy Expression

In [33]:
```
X, Y=symbols('X, Y')
YEq=Eq(Y, ~X); YEq
```

Out[33]:

$$Y = \neg X$$

In [34]:
```
TruthTabelGenrator(YEq)
```

Out[34]:

	X	Y
0	0	1
1	1	0

In [35]:
```
YEqN=lambdify(X, YEq.rhs, dummify=False)
YEqN(0), YEqN(1)
```

Out[35]:
```
(True, False)
```

7.2 myHDL Module

In [36]:
```
@block
def NotGate(X, Y):
    """
    NOT gate exsample module

    Input:
        X(bool): input

    Output:
        Y(bool): ouput

    """
    @always_comb
    def logic():
        Y.next=not X

    return instances()
```

7.3 myHDL Testing

In [37]:
```
#generate random test values for NotGate_TB
#stimules input In
TestLen=10
np.random.seed(14)
XTVs=np.random.randint(0,2, TestLen).astype(int)
XTVs
```

Out[37]:
```
array([1, 0, 0, 0, 1, 1, 0, 0, 0, 0])
```

In [38]:
```
Peeker.clear()
X=Signal(bool(0)); Peeker(X, 'X')
Y=Signal(bool(0)); Peeker(Y, 'Y')

DUT=NotGate(X, Y)

def NotGate_TB():
    """
    myHDL only testbench for module `NotGate`
    """

    @instance
    def stimules():
        for i in range(TestLen):
            X.next=int(XTVs[i])
            yield delay(1)

        raise StopSimulation()

    return instances()

sim=Simulation(DUT, NotGate_TB(), *Peeker.instances()).run()
```

In [39]:
```
Peeker.to_wavedrom()
```

In [40]:
```
NotData=Peeker.to_dataframe()
NotData
```

Out[40]:

	X	Y
0	1	0
1	0	1
4	1	0
6	0	1

In [41]:
```
NotData['YRef']=NotData['X'].apply(YEqN).astype(int)
NotData
```

Out[41]:

	X	Y	YRef
0	1	0	0
1	0	1	1
4	1	0	0
6	0	1	1

In [42]:
```
Test=(NotData['Y']==NotData['YRef']).all()
print(f'Module `NotGate` works as exspected: {Test}')
```

```
Module `NotGate` works as exspected: True
```

7.4 Verilog Conversion

In [43]:
```
DUT.convert()
VerilogTextReader('NotGate');
```

```
***Verilog modual from NotGate.v***

 // File: NotGate.v
// Generated by MyHDL 0.10
// Date: Mon Sep  3 14:44:38 2018


`timescale 1ns/10ps

module NotGate (
    X,
    Y
);
// NOT gate exsample module
//
// Input:
//     X(bool): input
//
// Output:
//     Y(bool): ouput

input X;
output Y;
wire Y;




assign Y = (!X);

endmodule
```

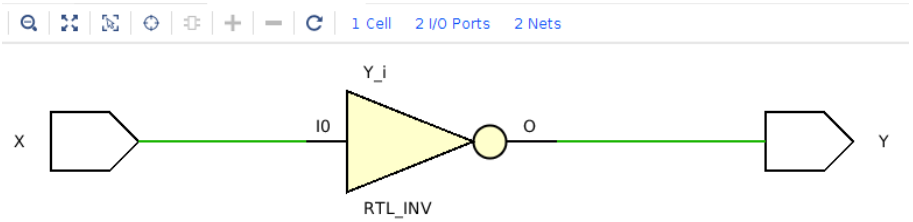


Figure 11: NotGate RTL schematic; Xilinx Vivado 2017.4

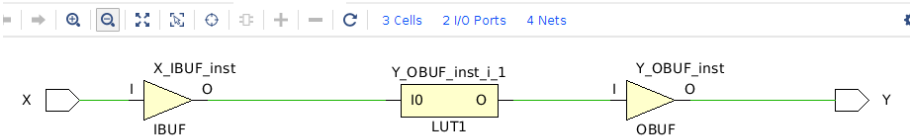


Figure 12: NotGate Synthesized Schematic; Xilinx Vivado 2017.4

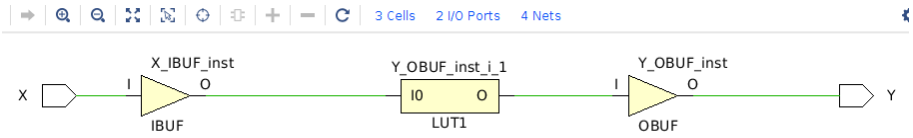


Figure 13: NotGate Implementated Schematic; Xilinx Vivado 2017.4

7.5 myHDL to Verilog Testbench

In [44]:
```
#create BitVector for NotGate_TBV
XTVs=intbv(int(''.join(XTVs.astype(str)), 2))[TestLen:]
XTVs, bin(XTVs)
```

Out[44]:
```
(intbv(560), '1000110000')
```

In [45]:

```
@block
def NotGate_TBV():
    """
    myHDL -> Verilog testbench for module `NotGate`
    """
    X=Signal(bool(0))
    Y=Signal(bool(0))

    @always_comb
    def print_data():
        print(X, Y)

    #Test Signal Bit Vector
    XTV=Signal(XTVs)

    DUT=NotGate(X, Y)


    @instance
    def stimules():
        for i in range(TestLen):
            X.next=XTV[i]
            yield delay(1)

        raise StopSimulation()
    return instances()

TB=NotGate_TBV()
TB.convert(hdl="Verilog", initial_values=True)
VerilogTextReader('NotGate_TBV');
```

```
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
***Verilog modual from NotGate_TBV.v***

 // File: NotGate_TBV.v
// Generated by MyHDL 0.10
// Date: Mon Sep  3 14:44:48 2018


`timescale 1ns/10ps

module NotGate_TBV (

);
// myHDL -> Verilog testbench for module `NotGate`


wire Y;
reg X = 0;
wire [9:0] XTV;

assign XTV = 10'd560;


always @(X, Y) begin: NOTGATE_TBV_PRINT_DATA
    $write("%h", X);
```

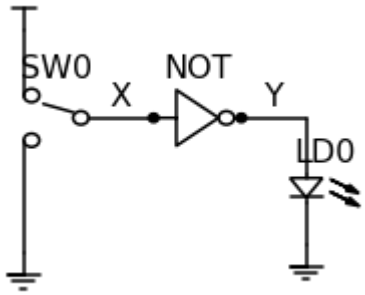
```
    $write(" ");
    $write("%h", Y);
    $write("\n");
end



assign Y = (!X);


initial begin: NOTGATE_TBV_STIMULES
    integer i;
    for (i=0; i<10; i=i+1) begin
        X <= XTV[i];
        # 1;
    end
    $finish;
end

endmodule


/home/iridium/anaconda3/lib/python3.6/site-packages/myhdl/conversion/_t
oVerilog.py:349: ToVerilogWarning: Signal is not driven: XTV
  category=ToVerilogWarning
```

7.6 PYNQ-Z1 Deployment

7.6.1 Board Circuit

In [46]:

```
d=schem.Drawing(unit=.5)
#add switch
SW0=d.add(e.SWITCH_SPDT2, reverse=True, label='SW0')

#connect to buffer
d.add(e.LINE, d='right', xy=SW0.a, l=d.unit*2, label='X')
d.add(e.DOT)
N0=d.add(l.NOT, label='NOT')

#buffer to LED
d.add(e.DOT)
d.add(e.LINE, d='right', l=d.unit*2, label='Y')
d.add(e.LINE, d='down', l=d.unit*2)
d.add(e.LED, rgtlabel='LD0', d='down')

#LED to ground
d.add(e.LINE, d='down', l=d.unit*1.5)
d.add(e.GND)

#swtich bottom to ground
d.add(e.LINE, d='down', xy=SW0.c, l=d.unit*4)
d.add(e.GND)


#switch top to rail
d.add(e.LINE, d='up', xy=SW0.b, l=d.unit*2.3)
d.add(e.VDD)
d.draw()
```



7.6.2 Board Constraint

In [47]:
```
ConstraintXDCTextReader('PYNQ_Z1Constraints_NotGate');
```

```
***Constraint file from PYNQ_Z1Constraints_NotGate.xdc***

 ## PYNQ-Z1 Board Constraints for NotGate.v
## Based on https://github.com/Digilent/digilent-xdc/blob/master/Arty-Master.xdc (https://github.com/Digilent/digilent-xdc/blob/master/Arty-Master.xdc)


## Switch 0
set_property -dict {PACKAGE_PIN M20 IOSTANDARD LVCMOS33} [get_ports
 {X}]


##LED 0
set_property -dict {PACKAGE_PIN R14 IOSTANDARD LVCMOS33} [get_ports
 {Y}]
```

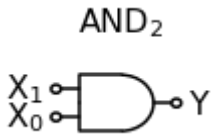
7.6.3 Video of Deployment

NOT Gate myHDL PYNQ-Z1 (YouTube (https://www.youtube.com/watch?v=5II5zOh6XAU))

8 AND

Definition 5 *An AND Gate is a logic gate the will yield a True response if and only if all of its inputs are also True. Its boolean expression is typically $$Y=X_0 \cdot X_1$$ using the multiplicative dot notation. Since if True is mapped to 1 and False is mapped 0, its output behavior is like that of multiplication of 1 and 0*

In [48]:
```
d = schem.Drawing(unit=.5)
G = d.add(l.AND2, label='$AND_2$')
d.add(e.DOT_OPEN, xy=G.out, rgtlabel='$Y$')
d.add(e.DOT_OPEN, xy=G.in1, lftlabel='$X_1$')
d.add(e.DOT_OPEN, xy=G.in2, lftlabel='$X_0$')
d.draw()
```



8.1 Sympy Expression

In [49]:
```
X0, X1, Y=symbols('X_0, X_1, Y')
YEq=Eq(Y, X0&X1); YEq
```

Out[49]:

$$Y = X_0 \wedge X_1$$

In [50]:
```
TruthTabelGenrator(YEq)
```

Out[50]:

	X_0	X_1	Y
0	0	0	0
1	0	1	0
2	1	0	0
3	1	1	1

In [51]:
```
YEqN=lambdify([X0, X1], YEq.rhs, dummify=False)
SystmaticVals=np.array(list(itertools.product([0,1], repeat=2)))
SystmaticVals, YEqN(SystmaticVals[:, 1], SystmaticVals[:, 0]).astype(ir
```

Out[51]:
```
(array([[0, 0],
        [0, 1],
        [1, 0],
        [1, 1]]), array([0, 0, 0, 1]))
```

8.2 myHDL Module

In [52]:
```
@block
def AndGate(X0, X1, Y):
    """
    And Gate demo module

    Input:
        X0(bool): And gate input 0
        X1(bool): And gate input 1

    Output:
        Y(bool): And gate ouput

    """
    @always_comb
    def logic():
        #note here that `and` is used since this
        #is a bit wise AND
        Y.next=X0 and X1

        #if `&` had been used the conversion
        #would yield `&` the .all() AND
        # when dealing with bus's and  behavior mux's this
        #distiction must be known
        #see:
        # https://stackoverflow.com/questions/17327680/what-is-the-diff

    return instances()
```

8.3 myHDL Testing

In [53]:
```
#generate systmatic and random test values for AndGate_TB
#stimules inputs X1 and X2
TestLen=10
SystmaticVals=list(itertools.product([0,1], repeat=2))
X0TVs=np.array([i[1] for i in SystmaticVals]).astype(int)
np.random.seed(15)
X0TVs=np.append(X0TVs, np.random.randint(0,2, TestLen)).astype(int)

X1TVs=np.array([i[0] for i in SystmaticVals]).astype(int)
#the random genrator must have a differint seed beween each generation
#call in order to produce differint values for each call
np.random.seed(16)
X1TVs=np.append(X1TVs, np.random.randint(0,2, TestLen)).astype(int)

TestLen=len(X1TVs)
SystmaticVals, X1TVs, X0TVs, TestLen
```

Out[53]:
```
([(0, 0), (0, 1), (1, 0), (1, 1)],
 array([0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0]),
 array([0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1]),
 14)
```

In [54]:
```
Peeker.clear()
X0=Signal(bool(0)); Peeker(X0, 'X0')
X1=Signal(bool(0)); Peeker(X1, 'X1')
Y=Signal(bool(0)); Peeker(Y, 'Y')

DUT=AndGate(X0, X1, Y)

def AndGate_TB():
    """
    myHDL only testbench for module `AndGate`
    """

    @instance
    def stimules():
        for i in range(TestLen):
            X0.next=int(X0TVs[i])
            X1.next=int(X1TVs[i])
            yield delay(1)

        raise StopSimulation()

    return instances()

sim=Simulation(DUT, AndGate_TB(), *Peeker.instances()).run()
```

In [55]:
```
Peeker.to_wavedrom('X1', 'X0', 'Y')
```

In [56]:
```
AndData=Peeker.to_dataframe()
AndData=AndData[['X1', 'X0', 'Y']]
AndData
```

Out[56]:

	X1	X0	Y
0	0	0	0
1	0	1	0
2	1	0	0
3	1	1	1
4	1	0	0
5	0	1	0
6	1	0	0
7	1	1	1
9	1	0	0
10	0	0	0
11	1	1	1
13	0	1	0

In [57]:
```
AndData['YRef']=AndData.apply(lambda row:YEqN(row['X0'], row['X1']), ax
AndData
```

Out[57]:

	X1	X0	Y	YRef
0	0	0	0	0
1	0	1	0	0
2	1	0	0	0
3	1	1	1	1
4	1	0	0	0
5	0	1	0	0
6	1	0	0	0
7	1	1	1	1
9	1	0	0	0
10	0	0	0	0
11	1	1	1	1
13	0	1	0	0

In [58]:
```
Test=(AndData['Y']==AndData['YRef']).all()
print(f'Module `AndGate` works as exspected: {Test}')
```

```
Module `AndGate` works as exspected: True
```

8.4 Verilog Conversion

In [59]:
```
DUT.convert()
VerilogTextReader('AndGate');
```

```
***Verilog modual from AndGate.v***

 // File: AndGate.v
// Generated by MyHDL 0.10
// Date: Mon Sep  3 14:45:30 2018


`timescale 1ns/10ps

module AndGate (
    X0,
    X1,
    Y
);
// And Gate demo module
//
// Input:
//     X0(bool): And gate input 0
//     X1(bool): And gate input 1
//
// Output:
//     Y(bool): And gate ouput

input X0;
input X1;
output Y;
wire Y;




assign Y = (X0 && X1);

endmodule
```

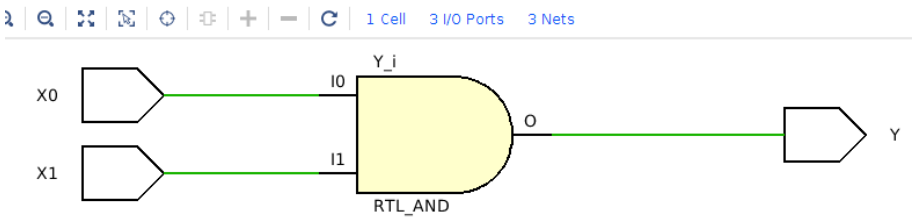


Figure 14: AndGate RTL schematic; Xilinx Vivado 2017.4

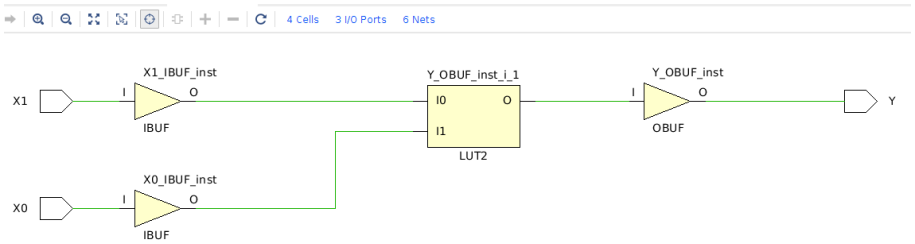


Figure 15: AndGate Synthesized Schematic; Xilinx Vivado 2017.4

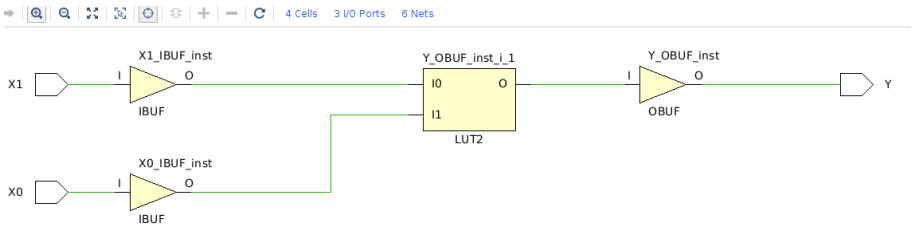


Figure 16: AndGate Implementated Schematic; Xilinx Vivado 2017.4

8.5 myHDL to Verilog Testbench

In [60]:
```
#create BitVectora for AndGate_TBV
X0TVs=intbv(int(''.join(X0TVs.astype(str)), 2))[TestLen:]
X1TVs=intbv(int(''.join(X1TVs.astype(str)), 2))[TestLen:]
X0TVs, bin(X0TVs), X1TVs, bin(X1TVs)
```

Out[60]:
```
(intbv(5479), '1010101100111', intbv(3830), '111011110110')
```

In [61]:

```
@block
def AndGate_TBV():
    """
    myHDL -> Verilog testbench for module `AndGate`
    """
    X0=Signal(bool(0))
    X1=Signal(bool(0))
    Y=Signal(bool(0))

    @always_comb
    def print_data():
        print(X0, X1, Y)

    #Test Signal Bit Vectors
    X0TV=Signal(X0TVs)
    X1TV=Signal(X1TVs)

    DUT=AndGate(X0, X1, Y)

    @instance
    def stimules():
        for i in range(TestLen):
            X0.next=int(X0TV[i])
            X1.next=int(X1TV[i])
            yield delay(1)

        raise StopSimulation()
    return instances()

TB=AndGate_TBV()
TB.convert(hdl="Verilog", initial_values=True)
VerilogTextReader('AndGate_TBV');
```

```
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
***Verilog modual from AndGate_TBV.v***

 // File: AndGate_TBV.v
// Generated by MyHDL 0.10
// Date: Mon Sep  3 14:45:40 2018


`timescale 1ns/10ps

module AndGate_TBV (

);
// myHDL -> Verilog testbench for module `AndGate`


wire Y;
reg X0 = 0;
reg X1 = 0;
wire [13:0] X0TV;
wire [13:0] X1TV;
```

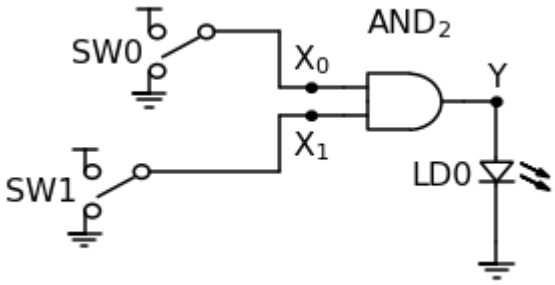
```
assign X0TV = 14'd5479;
assign X1TV = 14'd3830;


always @(X1, Y, X0) begin: ANDGATE_TBV_PRINT_DATA
    $write("%h", X0);
    $write(" ");
    $write("%h", X1);
    $write(" ");
    $write("%h", Y);
    $write("\n");
end



assign Y = (X0 && X1);


initial begin: ANDGATE_TBV_STIMULES
    integer i;
    for (i=0; i<14; i=i+1) begin
        X0 <= X0TV[i];
        X1 <= X1TV[i];
        # 1;
    end
    $finish;
end

endmodule


/home/iridium/anaconda3/lib/python3.6/site-packages/myhdl/conversion/_t
oVerilog.py:349: ToVerilogWarning: Signal is not driven: X0TV
  category=ToVerilogWarning
/home/iridium/anaconda3/lib/python3.6/site-packages/myhdl/conversion/_t
oVerilog.py:349: ToVerilogWarning: Signal is not driven: X1TV
  category=ToVerilogWarning
```

8.6 PYNQ-Z1 Deployment

8.6.1 Board Circuit

In [62]:

```
d=schem.Drawing(unit=.5)
#add elements
G=d.add(l.AND2,d='right', label='$AND_2$')

#Gate to led to gnd

d.add(e.LINE, d='right', xy=G.out)
d.add(e.DOT, label='$Y$')
d.add(e.LINE, d='down', l=d.unit*2)
LD0=d.add(e.LED, d='down', label='LD0')
d.add(e.LINE, d='down', l=d.unit*2)
d.add(e.GND)

d.add(e.LINE, d='left', xy=G.in1, l=d.unit)
d.add(e.DOT, label='$X_0$')
d.add(e.LINE,d='left', l=d.unit)
d.add(e.LINE,d='up', l=d.unit*2)
d.add(e.LINE,d='left', l=d.unit*2)
SW0=d.add(e.SWITCH_SPDT, lftlabel='SW0')
d.add(e.GND, xy=SW0.c)
d.add(e.VDD, xy=SW0.b)


d.add(e.LINE, d='left', xy=G.in2, l=d.unit)
d.add(e.DOT, botlabel='$X_1$')
d.add(e.LINE,d='left', l=d.unit)
d.add(e.LINE,d='down', l=d.unit*2)
d.add(e.LINE,d='left', l=d.unit*4)
SW1=d.add(e.SWITCH_SPDT, lftlabel='SW1')
d.add(e.GND, xy=SW1.c)
d.add(e.VDD, xy=SW1.b)
d.draw()
```



8.6.2 Board Constraint

In [63]:
```
ConstraintXDCTextReader('PYNQ_Z1Constraints_AndGate');
```

```
***Constraint file from PYNQ_Z1Constraints_AndGate.xdc***

 ## PYNQ-Z1 Board Constraints for AndGate.v
## Based on https://github.com/Digilent/digilent-xdc/blob/master/Arty-Master.xdc (https://github.com/Digilent/digilent-xdc/blob/master/Arty-Master.xdc)


## Switchs 0,1
set_property -dict {PACKAGE_PIN M20 IOSTANDARD LVCMOS33} [get_ports {X0}]
set_property -dict {PACKAGE_PIN M19 IOSTANDARD LVCMOS33} [get_ports {X1}]



##LED 0
set_property -dict {PACKAGE_PIN R14 IOSTANDARD LVCMOS33} [get_ports {Y}]
```

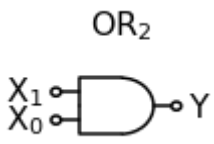
8.6.3 Video of Deployment

AND Gate myHDL PYNQ-Z1 (YouTube (https://www.youtube.com/watch?v=PqLGQezfr4U))

9 OR

Definition 6 *An OR Gate is a logic gate the will yield a True response if any of its inputs are True. Its boolean expression is typically $$Y=X_0 + X_1$$ using the addition notation. Since if True is mapped to 1 and False is mapped 0, its output behavior like that of the addition of 1 and 0 with the constraint of $1+1=1$ which is an axiom of Boolean Algebra.*

In [64]:
```
d = schem.Drawing(unit=.5)
G = d.add(l.AND2, label='$OR_2$')
d.add(e.DOT_OPEN, xy=G.out, rgtlabel='$Y$')
d.add(e.DOT_OPEN, xy=G.in1, lftlabel='$X_1$')
d.add(e.DOT_OPEN, xy=G.in2, lftlabel='$X_0$')
d.draw()
```



9.1 Sympy Expression

In [65]:
```
X0, X1, Y=symbols('X_0, X_1, Y')
YEq=Eq(Y, X0|X1); YEq
```

Out[65]:

$$Y = X_0 \vee X_1$$

In [66]:
```
TruthTabelGenrator(YEq)
```

Out[66]:

	X_0	X_1	Y
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	1

In [67]:
```
YEqN=lambdify([X0, X1], YEq.rhs, dummify=False)
SystmaticVals=np.array(list(itertools.product([0,1], repeat=2)))
SystmaticVals, YEqN(SystmaticVals[:, 1], SystmaticVals[:, 0]).astype(ir
```

Out[67]:
```
(array([[0, 0],
        [0, 1],
        [1, 0],
        [1, 1]]), array([0, 1, 1, 1]))
```

9.2 myHDL Module

In [68]:
```
@block
def OrGate(X0, X1, Y):
    """
    Or Gate demo module

    Input:
        X0(bool): Or gate input 0
        X1(bool): Or gate input 1

    Output:
        Y(bool): Or gate ouput

    """
    @always_comb
    def logic():
        #note here that `or` is used since this
        #is a bit wise OR
        Y.next=X0 or X1

        #if `|` had been used the conversion
        #would yield `||` the .all() OR
        # when dealing with bus's and  behavior mux's this
        #distiction must be known
        #see:
        # https://stackoverflow.com/questions/17327680/what-is-the-diff

    return instances()
```

9.3 myHDL Testing

In [69]:
```
#generate systmatic and random test values for OrGate_TB
#stimules inputs X1 and X2
TestLen=10
SystmaticVals=list(itertools.product([0,1], repeat=2))
X0TVs=np.array([i[1] for i in SystmaticVals]).astype(int)
np.random.seed(17)
X0TVs=np.append(X0TVs, np.random.randint(0,2, TestLen)).astype(int)

X1TVs=np.array([i[0] for i in SystmaticVals]).astype(int)
np.random.seed(18)
X1TVs=np.append(X1TVs, np.random.randint(0,2, TestLen)).astype(int)

TestLen=len(X1TVs)
SystmaticVals, X1TVs, X0TVs, TestLen
```

Out[69]:
```
([(0, 0), (0, 1), (1, 0), (1, 1)],
 array([0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0]),
 array([0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1]),
 14)
```

In [70]:

```
Peeker.clear()
X0=Signal(bool(0)); Peeker(X0, 'X0')
X1=Signal(bool(0)); Peeker(X1, 'X1')
Y=Signal(bool(0)); Peeker(Y, 'Y')

DUT=OrGate(X0, X1, Y)

def OrGate_TB():
    """
    myHDL only testbench for module `OrGate`
    """

    @instance
    def stimules():
        for i in range(TestLen):
            X0.next=int(X0TVs[i])
            X1.next=int(X1TVs[i])
            yield delay(1)

        raise StopSimulation()

    return instances()

sim=Simulation(DUT, OrGate_TB(), *Peeker.instances()).run()
```

In [71]:

```
Peeker.to_wavedrom('X1', 'X0', 'Y')
```

In [72]:
```
OrData=Peeker.to_dataframe()
OrData=OrData[['X1', 'X0', 'Y']]
OrData
```

Out[72]:

	X1	X0	Y
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	1
4	0	1	1
5	1	1	1
6	0	1	1
7	1	0	1
8	0	0	0
9	1	1	1
10	0	0	0
11	0	1	1
12	0	0	0
13	0	1	1

In [73]:
```
OrData['YRef']=OrData.apply(lambda row:YEqN(row['X0'], row['X1']), axis
OrData
```

Out[73]:

	X1	X0	Y	YRef
0	0	0	0	0
1	0	1	1	1
2	1	0	1	1
3	1	1	1	1
4	0	1	1	1
5	1	1	1	1
6	0	1	1	1
7	1	0	1	1
8	0	0	0	0
9	1	1	1	1
10	0	0	0	0
11	0	1	1	1
12	0	0	0	0
13	0	1	1	1

In [74]:
```
Test=(OrData['Y']==OrData['YRef']).all()
print(f'Module `OrGate` works as exspected: {Test}')
```

```
Module `OrGate` works as exspected: True
```

9.4 Verilog Conversion

In [75]:
```
DUT.convert()
VerilogTextReader('OrGate');
```

```
***Verilog modual from OrGate.v***

 // File: OrGate.v
// Generated by MyHDL 0.10
// Date: Mon Sep  3 14:46:15 2018


`timescale 1ns/10ps

module OrGate (
    X0,
    X1,
    Y
);
// Or Gate demo module
//
// Input:
//     X0(bool): Or gate input 0
//     X1(bool): Or gate input 1
//
// Output:
//     Y(bool): Or gate ouput

input X0;
input X1;
output Y;
wire Y;




assign Y = (X0 || X1);

endmodule
```

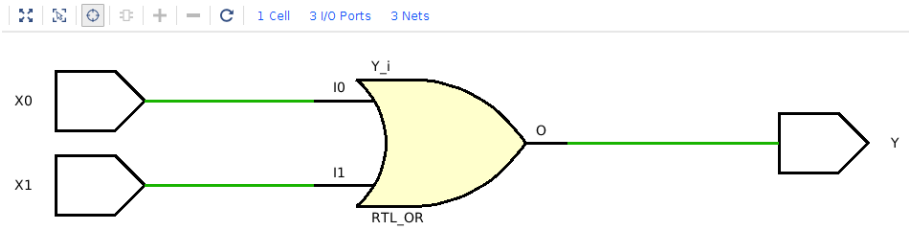


Figure 17: OrGate RTL schematic; Xilinx Vivado 2017.4

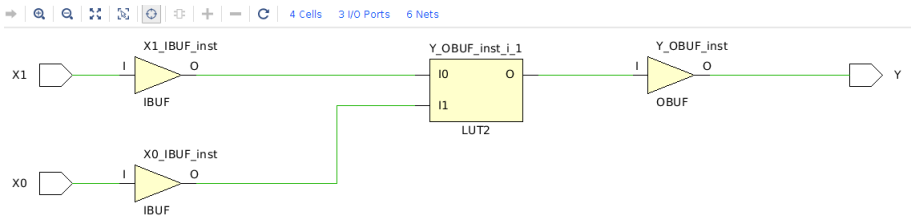


Figure 18: OrGate Synthesized Schematic; Xilinx Vivado 2017.4

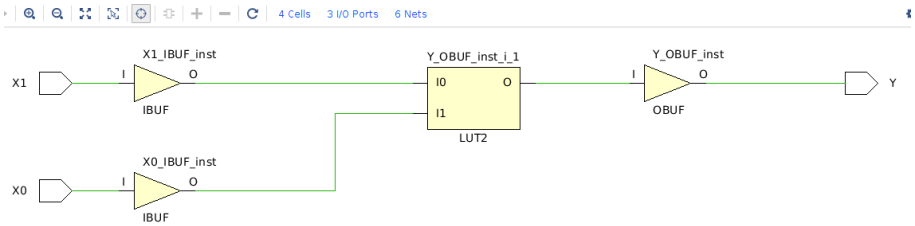


Figure 19: OrGate Implementated Schematic; Xilinx Vivado 2017.4

9.5 myHDL to Verilog Testbench

In [76]:
```
#create BitVectora for OrGate_TBV
X0TVs=intbv(int(''.join(X0TVs.astype(str)), 2))[TestLen:]
X1TVs=intbv(int(''.join(X1TVs.astype(str)), 2))[TestLen:]
X0TVs, bin(X0TVs), X1TVs, bin(X1TVs)
```

Out[76]:
```
(intbv(6037), '1011110010101', intbv(3408), '110101010000')
```

In [77]:

```
@block
def OrGate_TBV():
    """
    myHDL -> Verilog testbench for module `OrGate`
    """
    X0=Signal(bool(0))
    X1=Signal(bool(0))
    Y=Signal(bool(0))

    @always_comb
    def print_data():
        print(X0, X1, Y)

    #Test Signal Bit Vectors
    X0TV=Signal(X0TVs)
    X1TV=Signal(X1TVs)

    DUT=OrGate(X0, X1, Y)

    @instance
    def stimules():
        for i in range(TestLen):
            X0.next=int(X0TV[i])
            X1.next=int(X1TV[i])
            yield delay(1)

        raise StopSimulation()
    return instances()

TB=OrGate_TBV()
TB.convert(hdl="Verilog", initial_values=True)
VerilogTextReader('OrGate_TBV');
```

```
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
***Verilog modual from OrGate_TBV.v***

 // File: OrGate_TBV.v
// Generated by MyHDL 0.10
// Date: Mon Sep  3 14:46:26 2018


`timescale 1ns/10ps

module OrGate_TBV (

);
// myHDL -> Verilog testbench for module `OrGate`


wire Y;
reg X0 = 0;
reg X1 = 0;
wire [13:0] X0TV;
wire [13:0] X1TV;
```

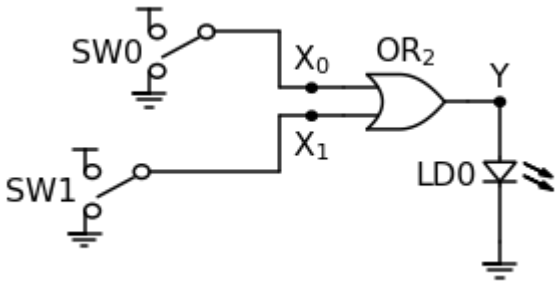
```
assign X0TV = 14'd6037;
assign X1TV = 14'd3408;


always @(X1, Y, X0) begin: ORGATE_TBV_PRINT_DATA
    $write("%h", X0);
    $write(" ");
    $write("%h", X1);
    $write(" ");
    $write("%h", Y);
    $write("\n");
end



assign Y = (X0 || X1);


initial begin: ORGATE_TBV_STIMULES
    integer i;
    for (i=0; i<14; i=i+1) begin
        X0 <= X0TV[i];
        X1 <= X1TV[i];
        # 1;
    end
    $finish;
end

endmodule


/home/iridium/anaconda3/lib/python3.6/site-packages/myhdl/conversion/_t
oVerilog.py:349: ToVerilogWarning: Signal is not driven: X0TV
  category=ToVerilogWarning
/home/iridium/anaconda3/lib/python3.6/site-packages/myhdl/conversion/_t
oVerilog.py:349: ToVerilogWarning: Signal is not driven: X1TV
  category=ToVerilogWarning
```

9.6 PYNQ-Z1 Deployment

9.6.1 Board Circuit

In [78]:

```
d=schem.Drawing(unit=.5)
#add elements
G=d.add(l.OR2,d='right', label='$OR_2$')

#Gate to led to gnd

d.add(e.LINE, d='right', xy=G.out)
d.add(e.DOT, label='$Y$')
d.add(e.LINE, d='down', l=d.unit*2)
LD0=d.add(e.LED, d='down', label='LD0')
d.add(e.LINE, d='down', l=d.unit*2)
d.add(e.GND)

d.add(e.LINE, d='left', xy=G.in1, l=d.unit)
d.add(e.DOT, label='$X_0$')
d.add(e.LINE,d='left', l=d.unit)
d.add(e.LINE,d='up', l=d.unit*2)
d.add(e.LINE,d='left', l=d.unit*2)
SW0=d.add(e.SWITCH_SPDT, lftlabel='SW0')
d.add(e.GND, xy=SW0.c)
d.add(e.VDD, xy=SW0.b)


d.add(e.LINE, d='left', xy=G.in2, l=d.unit)
d.add(e.DOT, botlabel='$X_1$')
d.add(e.LINE,d='left', l=d.unit)
d.add(e.LINE,d='down', l=d.unit*2)
d.add(e.LINE,d='left', l=d.unit*4)
SW1=d.add(e.SWITCH_SPDT, lftlabel='SW1')
d.add(e.GND, xy=SW1.c)
d.add(e.VDD, xy=SW1.b)
d.draw()
```



9.6.2 Board Constraint

In [79]:
```
ConstraintXDCTextReader('PYNQ_Z1Constraints_OrGate');
```

```
***Constraint file from PYNQ_Z1Constraints_OrGate.xdc***

 ## PYNQ-Z1 Board Constraints for XorGate.v
## Based on https://github.com/Digilent/digilent-xdc/blob/master/Arty-Master.xdc (https://github.com/Digilent/digilent-xdc/blob/master/Arty-Master.xdc)


## Switchs 0,1
set_property -dict {PACKAGE_PIN M20 IOSTANDARD LVCMOS33} [get_ports {X0}]
set_property -dict {PACKAGE_PIN M19 IOSTANDARD LVCMOS33} [get_ports {X1}]


##LED 0
set_property -dict {PACKAGE_PIN R14 IOSTANDARD LVCMOS33} [get_ports {Y}]
```

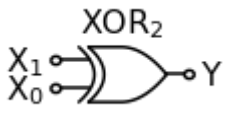
9.6.3 Video of Deployment

OR Gate myHDL PYNQ-Z1 (YouTube (https://www.youtube.com/watch?v=QKhMft17sac))

10 XOR

Definition 7 *An XOR Gate is a logic gate the will yield a True response if an only if only one of its inputs are True. Its boolean expression is typically $$Y=X_0 \oplus X_1$$ using its typical notation, wich is a refinment of the $+$ notation used for an OR Gate. But can also be expressed as $$Y=X_0\overline{X_1}+ \overline{X_0}X_1= (X_0+X_1) \cdot(\overline{X_0}+\overline{X_1})$$*

In [80]:
```
d = schem.Drawing(unit=.5)
G = d.add(l.XOR2, label='$XOR_2$')
d.add(e.DOT_OPEN, xy=G.out, rgtlabel='$Y$')
d.add(e.DOT_OPEN, xy=G.in1, lftlabel='$X_1$')
d.add(e.DOT_OPEN, xy=G.in2, lftlabel='$X_0$')
d.draw()
```



10.1 Sympy Expression

In [81]:
```
X0, X1, Y=symbols('X_0, X_1, Y')
YEq=Eq(Y, X0^X1); YEq
```

Out[81]:

$$Y = X_0 \veebar X_1$$

In [82]:
```
TruthTabelGenrator(YEq)
```

Out[82]:

	X_0	X_1	Y
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	0

In [83]:
```
#lampdfy must have maping manuly set from `Xor` to `np.bitwise_xor`
YEqN=lambdify([X0, X1], YEq.rhs, {'Xor':np.bitwise_xor}, dummify=False)
SystmaticVals=np.array(list(itertools.product([0,1], repeat=2)))
SystmaticVals, YEqN(SystmaticVals[:, 1], SystmaticVals[:, 0]).astype(ir
```

Out[83]:
```
(array([[0, 0],
        [0, 1],
        [1, 0],
        [1, 1]]), array([0, 1, 1, 0]))
```

10.2 myHDL Module

In [84]:
```
@block
def XorGate(X0, X1, Y):
    """
    XOR Gate demo module

    Input:
        X0(bool): XOR gate input 0
        X1(bool): XOR gate input 1

    Output:
        Y(bool): XOR gate ouput

    """
    @always_comb
    def logic():
        Y.next=X0 ^ X1

    return instances()
```

10.3 myHDL Testing

In [85]:
```
#generate systmatic and random test values for XorGate_TB
#stimules inputs X1 and X2
TestLen=10
SystmaticVals=list(itertools.product([0,1], repeat=2))
X0TVs=np.array([i[1] for i in SystmaticVals]).astype(int)
np.random.seed(19)
X0TVs=np.append(X0TVs, np.random.randint(0,2, TestLen)).astype(int)

X1TVs=np.array([i[0] for i in SystmaticVals]).astype(int)
np.random.seed(20)
X1TVs=np.append(X1TVs, np.random.randint(0,2, TestLen)).astype(int)

TestLen=len(X1TVs)
SystmaticVals, X1TVs, X0TVs, TestLen
```

Out[85]:
```
([(0, 0), (0, 1), (1, 0), (1, 1)],
 array([0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0]),
 array([0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0]),
 14)
```

In [86]:
```
Peeker.clear()
X0=Signal(bool(0)); Peeker(X0, 'X0')
X1=Signal(bool(0)); Peeker(X1, 'X1')
Y=Signal(bool(0)); Peeker(Y, 'Y')

DUT=XorGate(X0, X1, Y)

def XorGate_TB():
    """
    myHDL only testbench for module `XorGate`
    """

    @instance
    def stimules():
        for i in range(TestLen):
            X0.next=int(X0TVs[i])
            X1.next=int(X1TVs[i])
            yield delay(1)

        raise StopSimulation()

    return instances()

sim=Simulation(DUT, XorGate_TB(), *Peeker.instances()).run()
```

In [87]:
```
Peeker.to_wavedrom('X1', 'X0', 'Y')
```

In [88]:
```
XorData=Peeker.to_dataframe()
XorData=XorData[['X1', 'X0', 'Y']]
XorData
```

Out[88]:

	X1	X0	Y
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	0
5	0	0	0
6	1	1	0
7	1	0	1
8	0	0	0
9	0	1	1
10	1	0	1
11	0	0	0
12	1	1	0
13	0	0	0

In [89]:
```
XorData['YRef']=XorData.apply(lambda row:YEqN(row['X0'], row['X1']), ax
XorData
```

Out[89]:

	X1	X0	Y	YRef
0	0	0	0	0
1	0	1	1	1
2	1	0	1	1
3	1	1	0	0
5	0	0	0	0
6	1	1	0	0
7	1	0	1	1
8	0	0	0	0
9	0	1	1	1
10	1	0	1	1
11	0	0	0	0
12	1	1	0	0
13	0	0	0	0

In [90]:
```
Test=(XorData['Y']==XorData['YRef']).all()
print(f'Module `XorGate` works as exspected: {Test}')
```

```
Module `XorGate` works as exspected: True
```

10.4 Verilog Conversion

In [91]:
```
DUT.convert()
VerilogTextReader('XorGate');
```

```
***Verilog modual from XorGate.v***

 // File: XorGate.v
// Generated by MyHDL 0.10
// Date: Mon Sep  3 14:48:12 2018


`timescale 1ns/10ps

module XorGate (
    X0,
    X1,
    Y
);
// XOR Gate demo module
//
// Input:
//     X0(bool): XOR gate input 0
//     X1(bool): XOR gate input 1
//
// Output:
//     Y(bool): XOR gate ouput

input X0;
input X1;
output Y;
wire Y;




assign Y = (X0 ^ X1);

endmodule
```

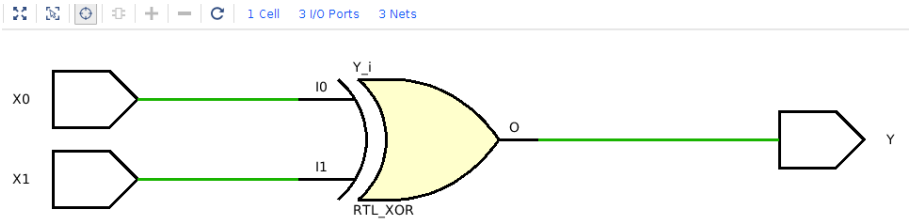


Figure 20: XorGate RTL schematic; Xilinx Vivado 2017.4

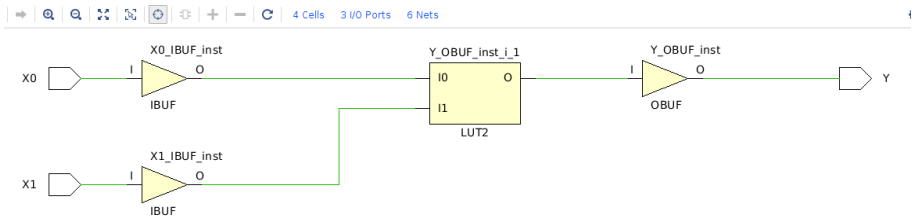


Figure 21: XorGate Synthesized Schematic; Xilinx Vivado 2017.4

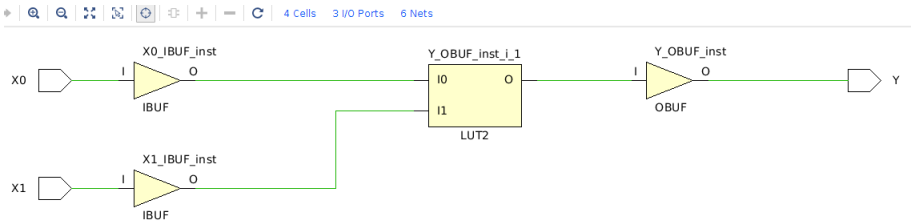


Figure 22: XorGate Implementated Schematic; Xilinx Vivado 2017.4

10.5 myHDL to Verilog Testbench

In [92]:
```
#create BitVectora for XorGate_TBV
X0TVs=intbv(int(''.join(X0TVs.astype(str)), 2))[TestLen:]
X1TVs=intbv(int(''.join(X1TVs.astype(str)), 2))[TestLen:]
X0TVs, bin(X0TVs), X1TVs, bin(X1TVs)
```

Out[92]:
```
(intbv(5778), '1011010010010', intbv(3786), '111011001010')
```

In [93]:

```
@block
def XorGate_TBV():
    """
    myHDL -> Verilog testbench for module `XorGate`
    """
    X0=Signal(bool(0))
    X1=Signal(bool(0))
    Y=Signal(bool(0))

    @always_comb
    def print_data():
        print(X0, X1, Y)

    #Test Signal Bit Vectors
    X0TV=Signal(X0TVs)
    X1TV=Signal(X1TVs)

    DUT=XorGate(X0, X1, Y)

    @instance
    def stimules():
        for i in range(TestLen):
            X0.next=int(X0TV[i])
            X1.next=int(X1TV[i])
            yield delay(1)

        raise StopSimulation()
    return instances()

TB=XorGate_TBV()
TB.convert(hdl="Verilog", initial_values=True)
VerilogTextReader('XorGate_TBV');
```

```
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
***Verilog modual from XorGate_TBV.v***

 // File: XorGate_TBV.v
// Generated by MyHDL 0.10
// Date: Mon Sep  3 14:48:26 2018


`timescale 1ns/10ps

module XorGate_TBV (

);
// myHDL -> Verilog testbench for module `XorGate`


wire Y;
reg X0 = 0;
reg X1 = 0;
wire [13:0] X0TV;
wire [13:0] X1TV;
```

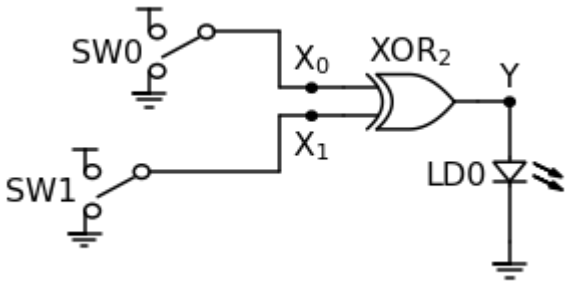
```
assign X0TV = 14'd5778;
assign X1TV = 14'd3786;


always @(X1, Y, X0) begin: XORGATE_TBV_PRINT_DATA
    $write("%h", X0);
    $write(" ");
    $write("%h", X1);
    $write(" ");
    $write("%h", Y);
    $write("\n");
end



assign Y = (X0 ^ X1);


initial begin: XORGATE_TBV_STIMULES
    integer i;
    for (i=0; i<14; i=i+1) begin
        X0 <= X0TV[i];
        X1 <= X1TV[i];
        # 1;
    end
    $finish;
end

endmodule


/home/iridium/anaconda3/lib/python3.6/site-packages/myhdl/conversion/_t
oVerilog.py:349: ToVerilogWarning: Signal is not driven: X0TV
  category=ToVerilogWarning
/home/iridium/anaconda3/lib/python3.6/site-packages/myhdl/conversion/_t
oVerilog.py:349: ToVerilogWarning: Signal is not driven: X1TV
  category=ToVerilogWarning
```

10.6 PYNQ-Z1 Deployment

10.6.1 Board Circuit

In [94]:

```
d=schem.Drawing(unit=.5)
#add elements
G=d.add(l.XOR2,d='right', label='$XOR_2$')

#Gate to led to gnd

d.add(e.LINE, d='right', xy=G.out)
d.add(e.DOT, label='$Y$')
d.add(e.LINE, d='down', l=d.unit*2)
LD0=d.add(e.LED, d='down', label='LD0')
d.add(e.LINE, d='down', l=d.unit*2)
d.add(e.GND)

d.add(e.LINE, d='left', xy=G.in1, l=d.unit)
d.add(e.DOT, label='$X_0$')
d.add(e.LINE,d='left', l=d.unit)
d.add(e.LINE,d='up', l=d.unit*2)
d.add(e.LINE,d='left', l=d.unit*2)
SW0=d.add(e.SWITCH_SPDT, lftlabel='SW0')
d.add(e.GND, xy=SW0.c)
d.add(e.VDD, xy=SW0.b)


d.add(e.LINE, d='left', xy=G.in2, l=d.unit)
d.add(e.DOT, botlabel='$X_1$')
d.add(e.LINE,d='left', l=d.unit)
d.add(e.LINE,d='down', l=d.unit*2)
d.add(e.LINE,d='left', l=d.unit*4)
SW1=d.add(e.SWITCH_SPDT, lftlabel='SW1')
d.add(e.GND, xy=SW1.c)
d.add(e.VDD, xy=SW1.b)
d.draw()
```



10.6.2 Board Constraint

In [95]:
```
ConstraintXDCTextReader('PYNQ_Z1Constraints_XorGate');
```

```
***Constraint file from PYNQ_Z1Constraints_XorGate.xdc***

 ## PYNQ-Z1 Board Constraints for OrGate.v
## Based on https://github.com/Digilent/digilent-xdc/blob/master/Arty-Master.xdc (https://github.com/Digilent/digilent-xdc/blob/master/Arty-Master.xdc)


## Switchs 0,1
set_property -dict {PACKAGE_PIN M20 IOSTANDARD LVCMOS33} [get_ports {X
0}]
set_property -dict {PACKAGE_PIN M19 IOSTANDARD LVCMOS33} [get_ports {X
1}]


##LED 0
set_property -dict {PACKAGE_PIN R14 IOSTANDARD LVCMOS33} [get_ports
 {Y}]
```

10.6.3 Video of Deployment

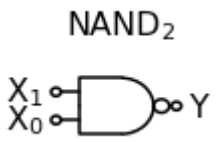
XOR Gate myHDL PYNQ-Z1 (YouTube (https://www.youtube.com/watch?v=Uys0Zhs7rlU))

11 NAND

Definition 8 *A NAND Gate is the negated, or the De Morgan equivalent, of an AND Gate. Such that is output will be True whenever there are False inputs to the gate. Its boolean expression is $$Y=\overline{X_0\cdot X_1}$$*

One peculiar aspect of the NAND Gate is that from a pure Boolean algebra position it is one of the two "Universal Gates". Which means that any other gate can be constructed from no more than NAND Gates. See https://en.wikipedia.org/wiki/NAND_logic#Making_other_gates_by_using_NAND_gates (https://en.wikipedia.org/wiki/NAND_logic#Making_other_gates_by_using_NAND_gates) for more info. And while this somewhat holds True for ASIC circuits, this not entirely true in terms of FPGAs

In [96]:
```
d = schem.Drawing(unit=.5)
G = d.add(l.NAND2, label='$NAND_2$')
d.add(e.DOT_OPEN, xy=G.out, rgtlabel='$Y$')
d.add(e.DOT_OPEN, xy=G.in1, lftlabel='$X_1$')
d.add(e.DOT_OPEN, xy=G.in2, lftlabel='$X_0$')
d.draw()
```



11.1 Sympy Expression

In [97]:
```
X0, X1, Y=symbols('X_0, X_1, Y')
YEq=Eq(Y, Nand(X0,X1)); YEq
```

Out[97]:

$$Y = \neg(X_0 \wedge X_1)$$

In [98]:
```
TruthTabelGenrator(YEq)
```

Out[98]:

	X_0	X_1	Y
0	0	0	1
1	0	1	1
2	1	0	1
3	1	1	0

In [99]:
```
YEqN=lambdify([X0, X1], YEq.rhs, dummify=False)
SystmaticVals=np.array(list(itertools.product([0,1], repeat=2)))
SystmaticVals, YEqN(SystmaticVals[:, 1], SystmaticVals[:, 0]).astype(ir
```

Out[99]:
```
(array([[0, 0],
        [0, 1],
        [1, 0],
        [1, 1]]), array([1, 1, 1, 0]))
```

11.2 myHDL Module

In [100]:
```
@block
def NAndGate(X0, X1, Y):
    """
    NAND Gate demo module

    Input:
        X0(bool): NAnd gate input 0
        X1(bool): NAnd gate input 1

    Output:
        Y(bool): NAnd gate ouput

    """
    @always_comb
    def logic():
        #note here that `and` is used since this
        #is a bit wise AND
        Y.next=not(X0 and X1)

        #if `&` had been used the conversion
        #would yield `&` the .all() AND
        # when dealing with bus's and  behavior mux's this
        #distiction must be known
        #see:
        # https://stackoverflow.com/questions/17327680/what-is-the-diff

    return instances()
```

11.3 myHDL Testing

In [101]:
```
#generate systmatic and random test values for AndGate_TB
#stimules inputs X1 and X2
TestLen=10
SystmaticVals=list(itertools.product([0,1], repeat=2))
X0TVs=np.array([i[1] for i in SystmaticVals]).astype(int)
np.random.seed(21)
X0TVs=np.append(X0TVs, np.random.randint(0,2, TestLen)).astype(int)

X1TVs=np.array([i[0] for i in SystmaticVals]).astype(int)
np.random.seed(23)
X1TVs=np.append(X1TVs, np.random.randint(0,2, TestLen)).astype(int)

TestLen=len(X1TVs)
SystmaticVals, X1TVs, X0TVs, TestLen
```

Out[101]:
```
([(0, 0), (0, 1), (1, 0), (1, 1)],
 array([0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1]),
 array([0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0]),
 14)
```

In [102]:
```
Peeker.clear()
X0=Signal(bool(0)); Peeker(X0, 'X0')
X1=Signal(bool(0)); Peeker(X1, 'X1')
Y=Signal(bool(0)); Peeker(Y, 'Y')

DUT=NAndGate(X0, X1, Y)

def NAndGate_TB():
    """
    myHDL only testbench for module `NAndGate`
    """

    @instance
    def stimules():
        for i in range(TestLen):
            X0.next=int(X0TVs[i])
            X1.next=int(X1TVs[i])
            yield delay(1)

        raise StopSimulation()

    return instances()

sim=Simulation(DUT, NAndGate_TB(), *Peeker.instances()).run()
```

In [103]:
```
Peeker.to_wavedrom('X1', 'X0', 'Y')
```

In [104]:
```
NAndData=Peeker.to_dataframe()
NAndData=NAndData[['X1', 'X0', 'Y']]
NAndData
```

Out[104]:

	X1	X0	Y
0	0	0	1
1	0	1	1
2	1	0	1
3	1	1	0
5	0	1	1
6	0	0	1
7	1	0	1
8	0	0	1
10	1	0	1
11	1	1	0
12	0	0	1
13	1	0	1

In [105]:
```
NAndData['YRef']=NAndData.apply(lambda row:YEqN(row['X0'], row['X1']),
NAndData
```

Out[105]:

	X1	X0	Y	YRef
0	0	0	1	1
1	0	1	1	1
2	1	0	1	1
3	1	1	0	0
5	0	1	1	1
6	0	0	1	1
7	1	0	1	1
8	0	0	1	1
10	1	0	1	1
11	1	1	0	0
12	0	0	1	1
13	1	0	1	1

In [106]:
```
Test=(NAndData['Y']==NAndData['YRef']).all()
print(f'Module `NAndGate` works as exspected: {Test}')
```

```
Module `NAndGate` works as exspected: True
```

11.4 Verilog Conversion

In [107]:
```
DUT.convert()
VerilogTextReader('NAndGate');
```

```
***Verilog modual from NAndGate.v***

 // File: NAndGate.v
// Generated by MyHDL 0.10
// Date: Mon Sep  3 14:49:01 2018


`timescale 1ns/10ps

module NAndGate (
    X0,
    X1,
    Y
);
// NAND Gate demo module
//
// Input:
//     X0(bool): NAnd gate input 0
//     X1(bool): NAnd gate input 1
//
// Output:
//     Y(bool): NAnd gate ouput

input X0;
input X1;
output Y;
wire Y;




assign Y = (!(X0 && X1));

endmodule
```

11.5 myHDL to Verilog Testbench

In [108]:
```
#create BitVectora for NAndGate_TBV
X0TVs=intbv(int(''.join(X0TVs.astype(str)), 2))[TestLen:]
X1TVs=intbv(int(''.join(X1TVs.astype(str)), 2))[TestLen:]
X0TVs, bin(X0TVs), X1TVs, bin(X1TVs)
```

Out[108]:
```
(intbv(5892), '1011100000100', intbv(3661), '111001001101')
```

In [109]:

```
@block
def NAndGate_TBV():
    """
    myHDL -> Verilog testbench for module `NAndGate`
    """
    X0=Signal(bool(0))
    X1=Signal(bool(0))
    Y=Signal(bool(0))

    @always_comb
    def print_data():
        print(X0, X1, Y)

    #Test Signal Bit Vectors
    X0TV=Signal(X0TVs)
    X1TV=Signal(X1TVs)

    DUT=NAndGate(X0, X1, Y)

    @instance
    def stimules():
        for i in range(TestLen):
            X0.next=int(X0TV[i])
            X1.next=int(X1TV[i])
            yield delay(1)

        raise StopSimulation()
    return instances()

TB=NAndGate_TBV()
TB.convert(hdl="Verilog", initial_values=True)
VerilogTextReader('NAndGate_TBV');
```

```
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
***Verilog modual from NAndGate_TBV.v***

 // File: NAndGate_TBV.v
// Generated by MyHDL 0.10
// Date: Mon Sep  3 14:49:07 2018


`timescale 1ns/10ps

module NAndGate_TBV (

);
// myHDL -> Verilog testbench for module `NAndGate`


wire Y;
reg X0 = 0;
reg X1 = 0;
wire [13:0] X0TV;
wire [13:0] X1TV;
```

```
assign X0TV = 14'd5892;
assign X1TV = 14'd3661;


always @(X1, Y, X0) begin: NANDGATE_TBV_PRINT_DATA
    $write("%h", X0);
    $write(" ");
    $write("%h", X1);
    $write(" ");
    $write("%h", Y);
    $write("\n");
end



assign Y = (!(X0 && X1));


initial begin: NANDGATE_TBV_STIMULES
    integer i;
    for (i=0; i<14; i=i+1) begin
        X0 <= X0TV[i];
        X1 <= X1TV[i];
        # 1;
    end
    $finish;
end

endmodule


/home/iridium/anaconda3/lib/python3.6/site-packages/myhdl/conversion/_t
oVerilog.py:349: ToVerilogWarning: Signal is not driven: X0TV
  category=ToVerilogWarning
/home/iridium/anaconda3/lib/python3.6/site-packages/myhdl/conversion/_t
oVerilog.py:349: ToVerilogWarning: Signal is not driven: X1TV
  category=ToVerilogWarning
```

Figure 23: NAndGate RTL schematic; Xilinx Vivado 2017.4

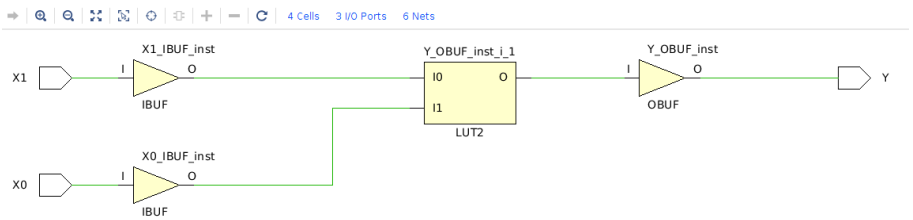


Figure 24: NAndGate Synthesized Schematic; Xilinx Vivado 2017.4

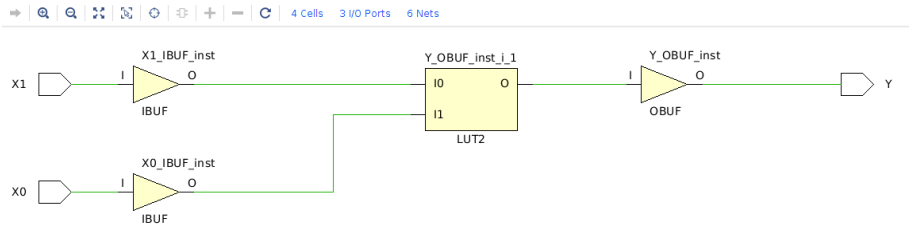


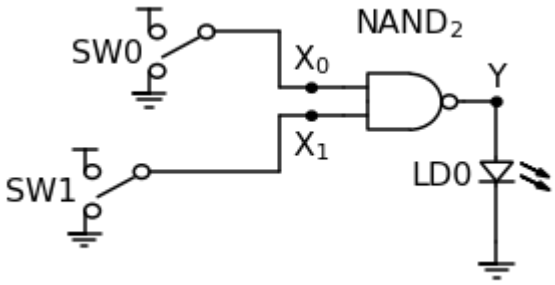
Figure 25: NAndGate Implementated Schematic; Xilinx Vivado 2017.4

11.6 PYNQ-Z1 Deyployment

11.6.1 Board Circuit

In [110]:

```
d=schem.Drawing(unit=.5)
#add elements
G=d.add(l.NAND2,d='right', label='$NAND_2$')

#Gate to led to gnd

d.add(e.LINE, d='right', xy=G.out)
d.add(e.DOT, label='$Y$')
d.add(e.LINE, d='down', l=d.unit*2)
LD0=d.add(e.LED, d='down', label='LD0')
d.add(e.LINE, d='down', l=d.unit*2)
d.add(e.GND)

d.add(e.LINE, d='left', xy=G.in1, l=d.unit)
d.add(e.DOT, label='$X_0$')
d.add(e.LINE,d='left', l=d.unit)
d.add(e.LINE,d='up', l=d.unit*2)
d.add(e.LINE,d='left', l=d.unit*2)
SW0=d.add(e.SWITCH_SPDT, lftlabel='SW0')
d.add(e.GND, xy=SW0.c)
d.add(e.VDD, xy=SW0.b)


d.add(e.LINE, d='left', xy=G.in2, l=d.unit)
d.add(e.DOT, botlabel='$X_1$')
d.add(e.LINE,d='left', l=d.unit)
d.add(e.LINE,d='down', l=d.unit*2)
d.add(e.LINE,d='left', l=d.unit*4)
SW1=d.add(e.SWITCH_SPDT, lftlabel='SW1')
d.add(e.GND, xy=SW1.c)
d.add(e.VDD, xy=SW1.b)
d.draw()
```



11.6.2 Board Constraint

In [111]:
```
ConstraintXDCTextReader('PYNQ_Z1Constraints_NAndGate');
```

```
***Constraint file from PYNQ_Z1Constraints_NAndGate.xdc***

 ## PYNQ-Z1 Board Constraints for NAndGate.v
## Based on https://github.com/Digilent/digilent-xdc/blob/master/Arty-Master.xdc (https://github.com/Digilent/digilent-xdc/blob/master/Arty-Master.xdc)


## Switchs 0,1
set_property -dict {PACKAGE_PIN M20 IOSTANDARD LVCMOS33} [get_ports {X
0}]
set_property -dict {PACKAGE_PIN M19 IOSTANDARD LVCMOS33} [get_ports {X
1}]


##LED 0
set_property -dict {PACKAGE_PIN R14 IOSTANDARD LVCMOS33} [get_ports
 {Y}]
```

11.6.3 Video of Deployment

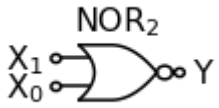
NAND Gate myHDL PYNQ-Z1 (YouTube (https://www.youtube.com/watch?v=Z1rnWfNapas))

12 NOR

Definition 9 *A NOR Gate is the negated, or the De Morgan equivalent, of an OR Gate. Such that is output will be True if and only if all its inputs are False. Its boolean expression is $$Y=\overline{X_0+ X_1}$$*

One peculiar aspect of the NOR Gate is that from a pure Boolean algebra position it is one of the two "Universal Gates". Which means that any other gate can be constructed from no more than NOR Gates. See https://en.wikipedia.org/wiki/NOR_logic#Making_other_gates_by_using_NOR_gates (https://en.wikipedia.org/wiki/NOR_logic#Making_other_gates_by_using_NOR_gates) for more info. And while this somewhat holds True for ASIC circuits, this not entirely true in terms of FPGAs

In [112]:
```
d = schem.Drawing(unit=.5)
G = d.add(l.NOR2, label='$NOR_2$')
d.add(e.DOT_OPEN, xy=G.out, rgtlabel='$Y$')
d.add(e.DOT_OPEN, xy=G.in1, lftlabel='$X_1$')
d.add(e.DOT_OPEN, xy=G.in2, lftlabel='$X_0$')
d.draw()
```



12.1 Sympy Expression

In [113]:
```
X0, X1, Y=symbols('X_0, X_1, Y')
YEq=Eq(Y, Nor(X0,X1)); YEq
```

Out[113]:

$$Y = \neg(X_0 \vee X_1)$$

In [114]:
```
TruthTabelGenrator(YEq)
```

Out[114]:

	X_0	X_1	Y
0	0	0	1
1	0	1	0
2	1	0	0
3	1	1	0

In [115]:
```
YEqN=lambdify([X0, X1], YEq.rhs, dummify=False)
SystmaticVals=np.array(list(itertools.product([0,1], repeat=2)))
SystmaticVals, YEqN(SystmaticVals[:, 1], SystmaticVals[:, 0]).astype(ir
```

Out[115]:
```
(array([[0, 0],
        [0, 1],
        [1, 0],
        [1, 1]]), array([1, 0, 0, 0]))
```

12.2 myHDL Module

In [116]:
```
@block
def NOrGate(X0, X1, Y):
    """
    NOr Gate demo module

    Input:
        X0(bool): Or gate input 0
        X1(bool): Or gate input 1

    Output:
        Y(bool): Or gate ouput

    """
    @always_comb
    def logic():
        #note here that `or` is used since this
        #is a bit wise OR
        Y.next=not(X0 or X1)

        #if `|` had been used the conversion
        #would yield `||` the .all() OR
        # when dealing with bus's and  behavior mux's this
        #distiction must be known
        #see:
        # https://stackoverflow.com/questions/17327680/what-is-the-diff

    return instances()
```

12.3 myHDL Testing

In [117]:
```
#generate systmatic and random test values for NOrGate_TB
#stimules inputs X1 and X2
TestLen=10
SystmaticVals=list(itertools.product([0,1], repeat=2))
X0TVs=np.array([i[1] for i in SystmaticVals]).astype(int)
np.random.seed(24)
X0TVs=np.append(X0TVs, np.random.randint(0,2, TestLen)).astype(int)

X1TVs=np.array([i[0] for i in SystmaticVals]).astype(int)
np.random.seed(25)
X1TVs=np.append(X1TVs, np.random.randint(0,2, TestLen)).astype(int)

TestLen=len(X1TVs)
SystmaticVals, X1TVs, X0TVs, TestLen
```

Out[117]:
```
([(0, 0), (0, 1), (1, 0), (1, 1)],
 array([0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0]),
 array([0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1]),
 14)
```

In [118]:
```
Peeker.clear()
X0=Signal(bool(0)); Peeker(X0, 'X0')
X1=Signal(bool(0)); Peeker(X1, 'X1')
Y=Signal(bool(0)); Peeker(Y, 'Y')

DUT=NOrGate(X0, X1, Y)

def NOrGate_TB():
    """
    myHDL only testbench for module `NOrGate`
    """

    @instance
    def stimules():
        for i in range(TestLen):
            X0.next=int(X0TVs[i])
            X1.next=int(X1TVs[i])
            yield delay(1)

        raise StopSimulation()

    return instances()

sim=Simulation(DUT, NOrGate_TB(), *Peeker.instances()).run()
```

In [119]:
```
Peeker.to_wavedrom('X1', 'X0', 'Y')
```

In [120]:
```
NOrData=Peeker.to_dataframe()
NOrData=NOrData[['X1', 'X0', 'Y']]
NOrData
```

Out[120]:

	X1	X0	Y
0	0	0	1
1	0	1	0
2	1	0	0
3	1	1	0
4	0	0	1
5	0	1	0
6	0	0	1
7	1	1	0
9	0	1	0
10	1	1	0
11	1	0	0
12	0	1	0

In [121]:
```
NOrData['YRef']=NOrData.apply(lambda row:YEqN(row['X0'], row['X1']), ax
NOrData
```

Out[121]:

	X1	X0	Y	YRef
0	0	0	1	1
1	0	1	0	0
2	1	0	0	0
3	1	1	0	0
4	0	0	1	1
5	0	1	0	0
6	0	0	1	1
7	1	1	0	0
9	0	1	0	0
10	1	1	0	0
11	1	0	0	0
12	0	1	0	0

In [122]:
```
Test=(NOrData['Y']==NOrData['YRef']).all()
print(f'Module `NOrGate` works as exspected: {Test}')
```

```
Module `NOrGate` works as exspected: True
```

12.4 Verilog Conversion

In [123]:
```
DUT.convert()
VerilogTextReader('NOrGate');
```

```
***Verilog modual from NOrGate.v***

 // File: NOrGate.v
// Generated by MyHDL 0.10
// Date: Mon Sep  3 14:49:57 2018


`timescale 1ns/10ps

module NOrGate (
    X0,
    X1,
    Y
);
// NOr Gate demo module
//
// Input:
//     X0(bool): Or gate input 0
//     X1(bool): Or gate input 1
//
// Output:
//     Y(bool): Or gate ouput

input X0;
input X1;
output Y;
wire Y;




assign Y = (!(X0 || X1));

endmodule
```

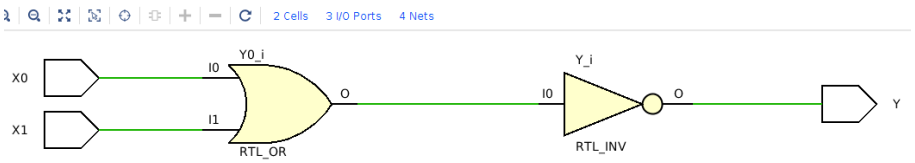


Figure 26: NOrGate RTL schematic; Xilinx Vivado 2017.4

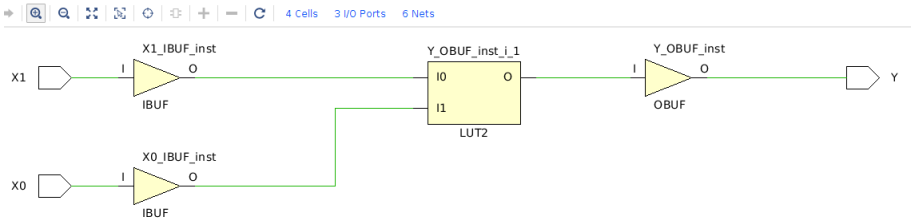


Figure 27: NOrGate Synthesized Schematic; Xilinx Vivado 2017.4

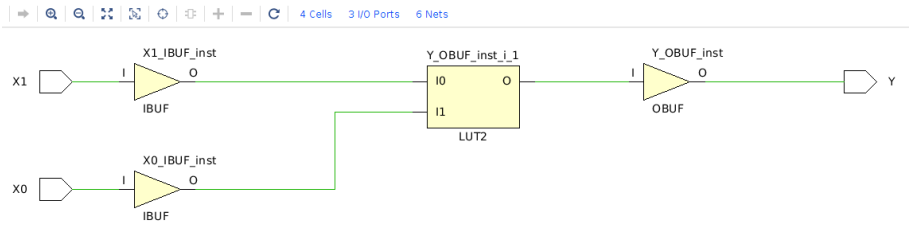


Figure 28: NOrGate Implementated Schematic; Xilinx Vivado 2017.4

12.5 myHDL to Verilog Testbench

In [124]:
```
#create BitVectora for NOrGate_TBV
X0TVs=intbv(int(''.join(X0TVs.astype(str)), 2))[TestLen:]
X1TVs=intbv(int(''.join(X1TVs.astype(str)), 2))[TestLen:]
X0TVs, bin(X0TVs), X1TVs, bin(X1TVs)
```

Out[124]:
```
(intbv(5499), '1010101111011', intbv(3180), '110001101100')
```

In [125]:

```
@block
def NOrGate_TBV():
    """
    myHDL -> Verilog testbench for module `NOrGate`
    """
    X0=Signal(bool(0))
    X1=Signal(bool(0))
    Y=Signal(bool(0))

    @always_comb
    def print_data():
        print(X0, X1, Y)

    #Test Signal Bit Vectors
    X0TV=Signal(X0TVs)
    X1TV=Signal(X1TVs)

    DUT=NOrGate(X0, X1, Y)

    @instance
    def stimules():
        for i in range(TestLen):
            X0.next=int(X0TV[i])
            X1.next=int(X1TV[i])
            yield delay(1)

        raise StopSimulation()
    return instances()

TB=NOrGate_TBV()
TB.convert(hdl="Verilog", initial_values=True)
VerilogTextReader('NOrGate_TBV');
```

```
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
***Verilog modual from NOrGate_TBV.v***

 // File: NOrGate_TBV.v
// Generated by MyHDL 0.10
// Date: Mon Sep  3 14:50:11 2018


`timescale 1ns/10ps

module NOrGate_TBV (

);
// myHDL -> Verilog testbench for module `NOrGate`


wire Y;
reg X0 = 0;
reg X1 = 0;
wire [13:0] X0TV;
wire [13:0] X1TV;
```

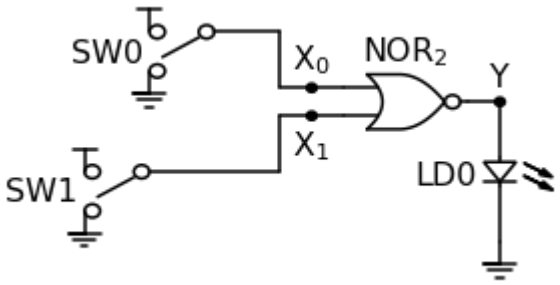
```
assign X0TV = 14'd5499;
assign X1TV = 14'd3180;


always @(X1, Y, X0) begin: NORGATE_TBV_PRINT_DATA
    $write("%h", X0);
    $write(" ");
    $write("%h", X1);
    $write(" ");
    $write("%h", Y);
    $write("\n");
end



assign Y = (!(X0 || X1));


initial begin: NORGATE_TBV_STIMULES
    integer i;
    for (i=0; i<14; i=i+1) begin
        X0 <= X0TV[i];
        X1 <= X1TV[i];
        # 1;
    end
    $finish;
end

endmodule


/home/iridium/anaconda3/lib/python3.6/site-packages/myhdl/conversion/_t
oVerilog.py:349: ToVerilogWarning: Signal is not driven: X0TV
  category=ToVerilogWarning
/home/iridium/anaconda3/lib/python3.6/site-packages/myhdl/conversion/_t
oVerilog.py:349: ToVerilogWarning: Signal is not driven: X1TV
  category=ToVerilogWarning
```

12.6 PYNQ-Z1 Deployment

12.6.1 Board Circuit

In [126]:

```
d=schem.Drawing(unit=.5)
#add elements
G=d.add(l.NOR2,d='right', label='$NOR_2$')

#Gate to led to gnd

d.add(e.LINE, d='right', xy=G.out)
d.add(e.DOT, label='$Y$')
d.add(e.LINE, d='down', l=d.unit*2)
LD0=d.add(e.LED, d='down', label='LD0')
d.add(e.LINE, d='down', l=d.unit*2)
d.add(e.GND)

d.add(e.LINE, d='left', xy=G.in1, l=d.unit)
d.add(e.DOT, label='$X_0$')
d.add(e.LINE,d='left', l=d.unit)
d.add(e.LINE,d='up', l=d.unit*2)
d.add(e.LINE,d='left', l=d.unit*2)
SW0=d.add(e.SWITCH_SPDT, lftlabel='SW0')
d.add(e.GND, xy=SW0.c)
d.add(e.VDD, xy=SW0.b)


d.add(e.LINE, d='left', xy=G.in2, l=d.unit)
d.add(e.DOT, botlabel='$X_1$')
d.add(e.LINE,d='left', l=d.unit)
d.add(e.LINE,d='down', l=d.unit*2)
d.add(e.LINE,d='left', l=d.unit*4)
SW1=d.add(e.SWITCH_SPDT, lftlabel='SW1')
d.add(e.GND, xy=SW1.c)
d.add(e.VDD, xy=SW1.b)
d.draw()
```



12.6.2 Board Constraint

In [127]:
```
ConstraintXDCTextReader('PYNQ_Z1Constraints_NOrGate');
```

```
***Constraint file from PYNQ_Z1Constraints_NOrGate.xdc***

 ## PYNQ-Z1 Board Constraints for NorGate.v
## Based on https://github.com/Digilent/digilent-xdc/blob/master/Arty-Master.xdc (https://github.com/Digilent/digilent-xdc/blob/master/Arty-Master.xdc)


## Switchs 0,1
set_property -dict {PACKAGE_PIN M20 IOSTANDARD LVCMOS33} [get_ports {X0}]
set_property -dict {PACKAGE_PIN M19 IOSTANDARD LVCMOS33} [get_ports {X1}]


##LED 0
set_property -dict {PACKAGE_PIN R14 IOSTANDARD LVCMOS33} [get_ports {Y}]
```

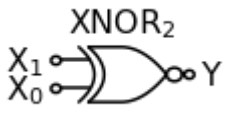
12.6.3 Video of Deployment

AND Gate myHDL PYNQ-Z1 (YouTube (https://www.youtube.com/watch?v=xtFvUx9fSAI))

13 XNOR

Definition 10 *An XNOR Gate is the negated, or the De Morgan equivalent, of an XOR Gate. Such that is output will be True if and only if all its inputs are the same. Its boolean expression is typically $$Y=X_0 \odot X_1$$ using its typical notation, wich is a refinment of the $\cdot$ notation used for an AND Gate. But can also be expressed as $$Y=X_0 X_1+ \overline{X_0}\overline{X_1}$$*

In [128]:
```
d = schem.Drawing(unit=.5)
G = d.add(l.XNOR2, label='$XNOR_2$')
d.add(e.DOT_OPEN, xy=G.out, rgtlabel='$Y$')
d.add(e.DOT_OPEN, xy=G.in1, lftlabel='$X_1$')
d.add(e.DOT_OPEN, xy=G.in2, lftlabel='$X_0$')
d.draw()
```



13.1 Sympy Expression

In [129]:
```
X0, X1, Y=symbols('X_0, X_1, Y')
#expanded exspresion of XNOR
YEq=Eq(Y, X0&X1|~X0&~X1); YEq
```

Out[129]:

$$Y = \left(X_0 \wedge X_1\right) \vee \left(\neg X_0 \wedge \neg X_1\right)$$

In [130]:
```
TruthTabelGenrator(YEq)
```

Out[130]:

	X_0	X_1	Y
0	0	0	1
1	0	1	0
2	1	0	0
3	1	1	1

In [131]:
```
YEqN=lambdify([X0, X1], YEq.rhs, dummify=False)
SystmaticVals=np.array(list(itertools.product([0,1], repeat=2)))
SystmaticVals, YEqN(SystmaticVals[:, 1], SystmaticVals[:, 0]).astype(ir
```

Out[131]:
```
(array([[0, 0],
        [0, 1],
        [1, 0],
        [1, 1]]), array([1, 0, 0, 1]))
```

13.2 myHDL Module

In [132]:
```
@block
def XNOrGate(X0, X1, Y):
    """
    XNOR Gate demo module

    Input:
        X0(bool): XOR gate input 0
        X1(bool): XOR gate input 1

    Output:
        Y(bool): XOR gate ouput

    """
    @always_comb
    def logic():
        Y.next=not(X0 ^ X1)

    return instances()
```

13.3 myHDL Testing

In [133]:
```
#generate systmatic and random test values for XNOrGate_TB
#stimules inputs X1 and X2
TestLen=10
SystmaticVals=list(itertools.product([0,1], repeat=2))
X0TVs=np.array([i[1] for i in SystmaticVals]).astype(int)
np.random.seed(26)
X0TVs=np.append(X0TVs, np.random.randint(0,2, TestLen)).astype(int)

X1TVs=np.array([i[0] for i in SystmaticVals]).astype(int)
np.random.seed(27)
X1TVs=np.append(X1TVs, np.random.randint(0,2, TestLen)).astype(int)

TestLen=len(X1TVs)
SystmaticVals, X1TVs, X0TVs, TestLen
```

Out[133]:
```
([(0, 0), (0, 1), (1, 0), (1, 1)],
 array([0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1]),
 array([0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0]),
 14)
```

In [134]:
```
Peeker.clear()
X0=Signal(bool(0)); Peeker(X0, 'X0')
X1=Signal(bool(0)); Peeker(X1, 'X1')
Y=Signal(bool(0)); Peeker(Y, 'Y')

DUT=XNOrGate(X0, X1, Y)

def XNOrGate_TB():
    """
    myHDL only testbench for module `NOrGate`
    """

    @instance
    def stimules():
        for i in range(TestLen):
            X0.next=int(X0TVs[i])
            X1.next=int(X1TVs[i])
            yield delay(1)

        raise StopSimulation()

    return instances()

sim=Simulation(DUT, XNOrGate_TB(), *Peeker.instances()).run()
```

In [135]:
```
Peeker.to_wavedrom('X1', 'X0', 'Y')
```

In [136]:
```
XNOrData=Peeker.to_dataframe()
XNOrData=XNOrData[['X1', 'X0', 'Y']]
XNOrData
```

Out[136]:

	X1	X0	Y
0	0	0	1
1	0	1	0
2	1	0	0
3	1	1	1
5	0	0	1
7	1	0	0
8	0	1	0
9	0	0	1
10	1	1	1
11	0	1	0
12	1	0	0

In [137]:
```
XNOrData['YRef']=XNOrData.apply(lambda row:YEqN(row['X0'], row['X1']),
XNOrData
```

Out[137]:

	X1	X0	Y	YRef
0	0	0	1	1
1	0	1	0	0
2	1	0	0	0
3	1	1	1	1
5	0	0	1	1
7	1	0	0	0
8	0	1	0	0
9	0	0	1	1
10	1	1	1	1
11	0	1	0	0
12	1	0	0	0

In [138]:
```
Test=(XNOrData['Y']==XNOrData['YRef']).all()
print(f'Module `XNOrGate` works as exspected: {Test}')
```

```
Module `XNOrGate` works as exspected: True
```

13.4 Verilog Conversion

In [139]:
```
DUT.convert()
VerilogTextReader('XNOrGate');
```

```
***Verilog modual from XNOrGate.v***

 // File: XNOrGate.v
// Generated by MyHDL 0.10
// Date: Mon Sep  3 14:50:48 2018


`timescale 1ns/10ps

module XNOrGate (
    X0,
    X1,
    Y
);
// XNOR Gate demo module
//
// Input:
//     X0(bool): XOR gate input 0
//     X1(bool): XOR gate input 1
//
// Output:
//     Y(bool): XOR gate ouput

input X0;
input X1;
output Y;
wire Y;




assign Y = (!(X0 ^ X1));

endmodule
```

Figure 29: XNOrGate RTL schematic; Xilinx Vivado 2017.4

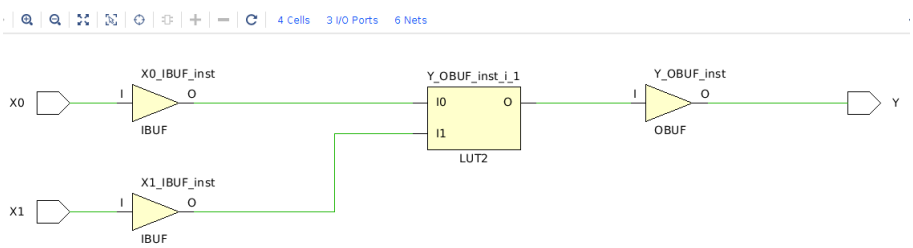


Figure 30: XNOrGate Synthesized Schematic; Xilinx Vivado 2017.4

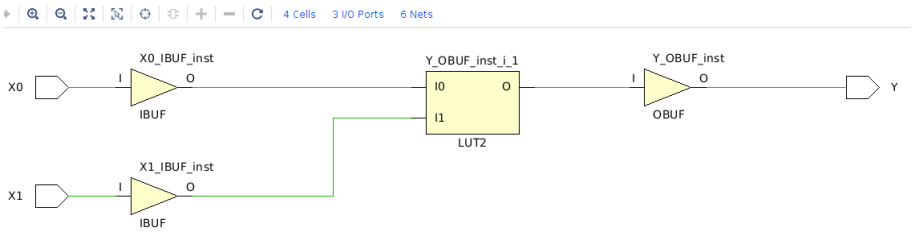


Figure 31: XNOrGate Implementated Schematic; Xilinx Vivado 2017.4

13.5 myHDL to Verilog Testbench

In [140]:
```
#create BitVectora for XNOrGate_TBV
X0TVs=intbv(int(''.join(X0TVs.astype(str)), 2))[TestLen:]
X1TVs=intbv(int(''.join(X1TVs.astype(str)), 2))[TestLen:]
X0TVs, bin(X0TVs), X1TVs, bin(X1TVs)
```

Out[140]:
```
(intbv(5676), '1011000101100', intbv(3659), '111001001011')
```

In [141]:

```
@block
def XNOrGate_TBV():
    """
    myHDL -> Verilog testbench for module `NOrGate`
    """
    X0=Signal(bool(0))
    X1=Signal(bool(0))
    Y=Signal(bool(0))

    @always_comb
    def print_data():
        print(X0, X1, Y)

    #Test Signal Bit Vectors
    X0TV=Signal(X0TVs)
    X1TV=Signal(X1TVs)

    DUT=XNOrGate(X0, X1, Y)

    @instance
    def stimules():
        for i in range(TestLen):
            X0.next=int(X0TV[i])
            X1.next=int(X1TV[i])
            yield delay(1)

        raise StopSimulation()
    return instances()

TB=XNOrGate_TBV()
TB.convert(hdl="Verilog", initial_values=True)
VerilogTextReader('XNOrGate_TBV');
```

```
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
***Verilog modual from XNOrGate_TBV.v***

 // File: XNOrGate_TBV.v
// Generated by MyHDL 0.10
// Date: Mon Sep  3 14:51:02 2018


`timescale 1ns/10ps

module XNOrGate_TBV (

);
// myHDL -> Verilog testbench for module `NOrGate`


wire Y;
reg X0 = 0;
reg X1 = 0;
wire [13:0] X0TV;
wire [13:0] X1TV;
```

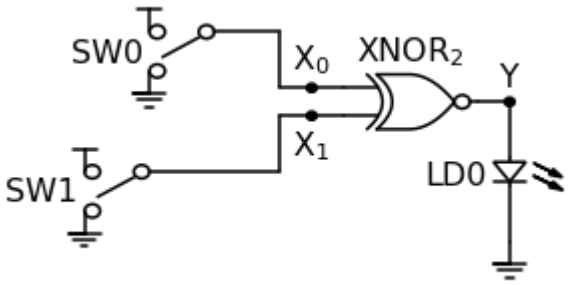
```
assign X0TV = 14'd5676;
assign X1TV = 14'd3659;


always @(X1, Y, X0) begin: XNORGATE_TBV_PRINT_DATA
    $write("%h", X0);
    $write(" ");
    $write("%h", X1);
    $write(" ");
    $write("%h", Y);
    $write("\n");
end



assign Y = (!(X0 ^ X1));


initial begin: XNORGATE_TBV_STIMULES
    integer i;
    for (i=0; i<14; i=i+1) begin
        X0 <= X0TV[i];
        X1 <= X1TV[i];
        # 1;
    end
    $finish;
end

endmodule


/home/iridium/anaconda3/lib/python3.6/site-packages/myhdl/conversion/_t
oVerilog.py:349: ToVerilogWarning: Signal is not driven: X0TV
  category=ToVerilogWarning
/home/iridium/anaconda3/lib/python3.6/site-packages/myhdl/conversion/_t
oVerilog.py:349: ToVerilogWarning: Signal is not driven: X1TV
  category=ToVerilogWarning
```

13.6 PYNQ-Z1 Deployment

13.6.1 Board Circuit

In [142]:

```
d=schem.Drawing(unit=.5)
#add elements
G=d.add(l.XNOR2,d='right', label='$XNOR_2$')

#Gate to led to gnd

d.add(e.LINE, d='right', xy=G.out)
d.add(e.DOT, label='$Y$')
d.add(e.LINE, d='down', l=d.unit*2)
LD0=d.add(e.LED, d='down', label='LD0')
d.add(e.LINE, d='down', l=d.unit*2)
d.add(e.GND)

d.add(e.LINE, d='left', xy=G.in1, l=d.unit)
d.add(e.DOT, label='$X_0$')
d.add(e.LINE,d='left', l=d.unit)
d.add(e.LINE,d='up', l=d.unit*2)
d.add(e.LINE,d='left', l=d.unit*2)
SW0=d.add(e.SWITCH_SPDT, lftlabel='SW0')
d.add(e.GND, xy=SW0.c)
d.add(e.VDD, xy=SW0.b)


d.add(e.LINE, d='left', xy=G.in2, l=d.unit)
d.add(e.DOT, botlabel='$X_1$')
d.add(e.LINE,d='left', l=d.unit)
d.add(e.LINE,d='down', l=d.unit*2)
d.add(e.LINE,d='left', l=d.unit*4)
SW1=d.add(e.SWITCH_SPDT, lftlabel='SW1')
d.add(e.GND, xy=SW1.c)
d.add(e.VDD, xy=SW1.b)
d.draw()
```



13.6.2 Board Constraint

In [143]:
```
ConstraintXDCTextReader('PYNQ_Z1Constraints_XNOrGate');
```

```
***Constraint file from PYNQ_Z1Constraints_XNOrGate.xdc***

 ## PYNQ-Z1 Board Constraints for XNOrGate.v
## Based on https://github.com/Digilent/digilent-xdc/blob/master/Arty-Master.xdc (https://github.com/Digilent/digilent-xdc/blob/master/Arty-Master.xdc)


## Switchs 0,1
set_property -dict {PACKAGE_PIN M20 IOSTANDARD LVCMOS33} [get_ports {X0}]
set_property -dict {PACKAGE_PIN M19 IOSTANDARD LVCMOS33} [get_ports {X1}]


##LED 0
set_property -dict {PACKAGE_PIN R14 IOSTANDARD LVCMOS33} [get_ports
 {Y}]
```

13.6.3 Video of Deployment

AND Gate myHDL PYNQ-Z1 (YouTube (https://www.youtube.com/watch?v=Uys0Zhs7rlU))

In [ ]:
```

```