

myHDL Combinational Logic Elements: Multiplexers (MUXs))

Steven K Armour

Table of Contents

- [1 Refrances](#)
- [2 Libraries and Helper functions](#)
- [3 Multiplexers](#)
- [4 2 Channel Input:1 Channel Output multiplexer in Gate Level Logic](#)
 - [4.1 Sympy Expression](#)
 - [4.2 myHDL Module](#)
 - [4.3 myHDL Testing](#)
 - [4.4 Verilog Conversion](#)
 - [4.5 myHDL to Verilog Testbench](#)
 - [4.6 PYNQ-Z1 Deployment](#)
 - [4.6.1 Board Circuit](#)
 - [4.6.2 Board Constraint](#)
 - [4.6.3 Video of Deployment](#)
- [5 4 Channel Input : 1 Channel Output multiplexer in Gate Level Logic](#)
 - [5.1 Sympy Expression](#)
 - [5.2 myHDL Module](#)
 - [5.3 myHDL Testing](#)
 - [5.4 Verilog Conversion](#)
 - [5.5 myHDL to Verilog Testbench](#)
 - [5.6 PYNQ-Z1 Deployment](#)
 - [5.6.1 Board Circuit](#)
 - [5.6.2 Board Constraint](#)
 - [5.6.3 Video of Deployment](#)
- [6 Shannon's Expansion Formula & Stacking of MUXs](#)
- [7 4 Channel Input: 1 Channel Output multiplexer via MUX Stacking](#)

▼ 1 Refrances

@misc{xu_2018, title={Introduction to Digital Systems Supplementary Reading Shannon's Expansion Formulas and Compressed Truth Table}, author={Xu, Xuping}, year={Fall 2017} site=<http://ecse.bd.psu.edu/cse271/comprttb.pdf> (<http://ecse.bd.psu.edu/cse271/comprttb.pdf>). }

▼ 2 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  
  
#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	0.23.4
matplotlib	2.1.0
sympy	1.3
itertools	The 'itertools' distribution was not found and is required by the application
SchemDraw	0.3.0

Sat Sep 22 02:10:44 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 boolean expression
▼     BoolSymFunc: sympy boolean 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

```

▼ 3 Multiplexers

Definition 1 A Multiplexer, typically referred to as a MUX, is a Digital(or analog) switching unit that picks one input channel to be streamed to an output via a control input. For single output MUXs with 2^n inputs, there are then n input selection signals that make up the control work to select the input channel for output.

From a behavioral standpoint, a MUX can be thought of as an element that performs the same functionality as the *if-elif-else* (case) control statements found in almost every software language.

▼ 4 2 Channel Input:1 Channel Output multiplexer in Gate Level Logic

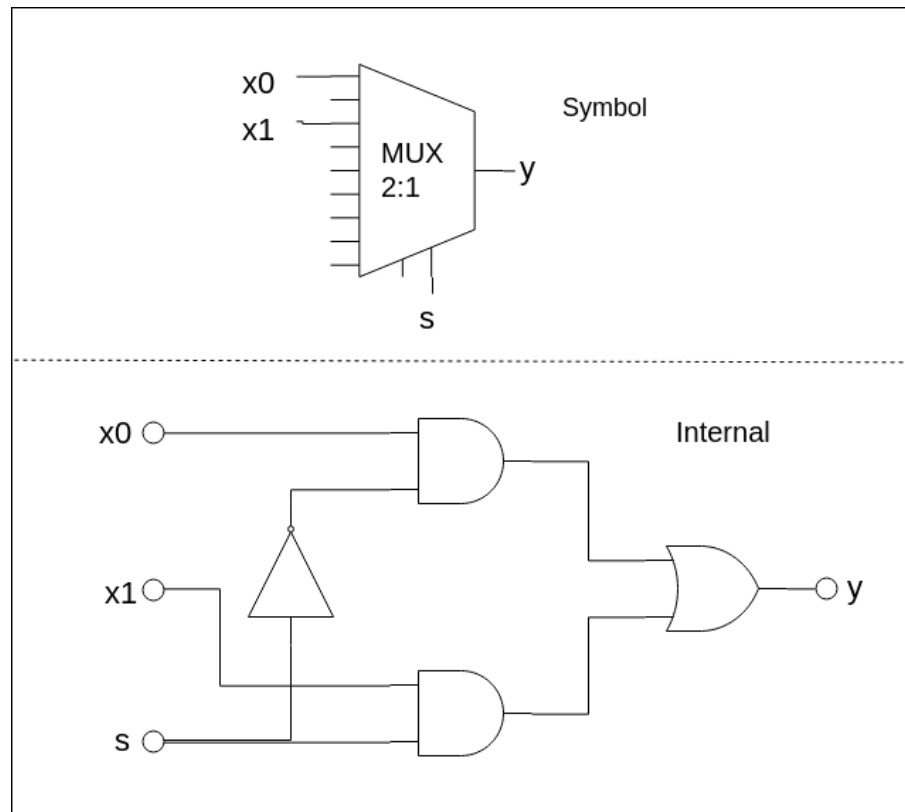


Figure 1: 2:1 MUX Symbol and Gate internals

▼ 4.1 Sympy Expression

```
In [4]: x0, x1, s, y=symbols('x0, x1, s, y')
        y2lEq=Eq(y, (~s&x0) |(s&x1) ); y2lEq
```

```
Out[4]:  $y = (s \wedge x_1) \vee (x_0 \wedge \neg s)$ 
```

```
In [5]: TruthTabelGenrator(y21Eq)[[x1, x0, s, y]]
```

```
Out[5]:
```

	x1	x0	s	y
0	0	0	0	0
1	1	0	0	0
2	0	1	0	1
3	1	1	0	1
4	0	0	1	0
5	1	0	1	1
6	0	1	1	0
7	1	1	1	1

```
In [6]: y21EqN=lambdify([x0, x1, s], y21Eq.rhs, dummify=False)
        SystmaticVals=np.array(list(itertools.product([0,1], repeat=3)))
        print(SystmaticVals)
        y21EqN(SystmaticVals[:, 1], SystmaticVals[:, 2], SystmaticVals[:, 0]).a
```

```
[[0 0 0]
 [0 0 1]
 [0 1 0]
 [0 1 1]
 [1 0 0]
 [1 0 1]
 [1 1 0]
 [1 1 1]]
```

```
Out[6]: array([0, 0, 1, 1, 0, 1, 0, 1])
```

▼ 4.2 myHDL Module

```
In [7]: @block
        ▼ def MUX2_1_Combo(x0, x1, s, y):
            """
            2:1 Multiplexer written in full combo
            Input:
            ▼ x0(bool): input channel 0
              x1(bool): input channel 1
              s(bool): channel selection input
            Output:
            ▼ y(bool): ouput
            """

            @always_comb
            ▼ def logic():
                y.next= (not s and x0) |(s and x1)

            return instances()
```

▼ 4.3 myHDL Testing

```
In [8]: ▼ #generate systmatic and random test values  
#stimules inputs X1 and X2  
TestLen=10  
SystmaticVals=list(itertools.product([0,1], repeat=3))  
  
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[2] for i in SystmaticVals]).astype(int)  
#the random genrator must have a differint seed between 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)  
  
sTVs=np.array([i[0] for i in SystmaticVals]).astype(int)  
#the random genrator must have a differint seed between each generation  
#call in order to produce differint values for each call  
np.random.seed(17)  
sTVs=np.append(sTVs, np.random.randint(0,2, TestLen)).astype(int)  
  
TestLen=len(x0TVs)  
x0TVs, x1TVs, sTVs, TestLen
```

```
Out[8]: (array([0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1]),  
array([0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0]),  
array([0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1]),  
18)
```

```

In [9]: Peeker.clear()
x0=Signal(bool(0)); Peeker(x0, 'x0')
x1=Signal(bool(0)); Peeker(x1, 'x1')
s=Signal(bool(0)); Peeker(s, 's')
y=Signal(bool(0)); Peeker(y, 'y')

DUT=MUX2_1_Combo(x0, x1, s, y)

▼ def MUX2_1_Combo_TB():
    """
    myHDL only testbench for module `MUX2_1_Combo`
    """

    @instance
    ▼ def stimules():
        ▼ for i in range(TestLen):
            x0.next=int(x0TVs[i])
            x1.next=int(x1TVs[i])
            s.next=int(sTVs[i])

            yield delay(1)

            raise StopSimulation()

        return instances()

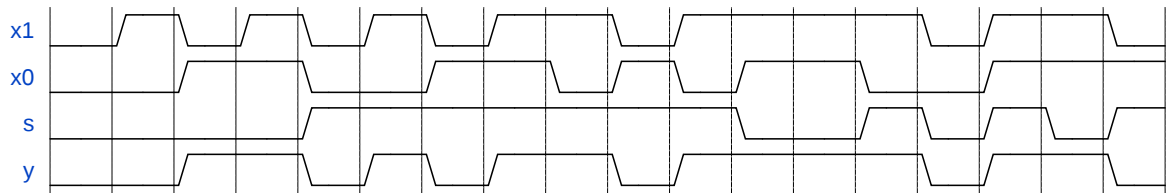
sim=Simulation(DUT, MUX2_1_Combo_TB(), *Peeker.instances()).run()

```

```

In [11]: Peeker.to_wavedrom('x1', 'x0', 's', 'y')

```



```
In [12]: MUX2_1_ComboData=Peeker.to_dataframe()  
MUX2_1_ComboData=MUX2_1_ComboData[['x1', 'x0', 's', 'y']]  
MUX2_1_ComboData
```

Out[12]:

	x1	x0	s	y
0	0	0	0	0
1	1	0	0	0
2	0	1	0	1
3	1	1	0	1
4	0	0	1	0
5	1	0	1	1
6	0	1	1	0
7	1	1	1	1
8	1	0	1	1
9	0	1	1	0
10	1	0	1	1

```
In [13]: MUX2_1_ComboData['yRef']=MUX2_1_ComboData.apply(lambda row:y21EqN(row['x1', 'x0', 's', 'y']),axis=1)  
MUX2_1_ComboData
```

Out[13]:

	x1	x0	s	y	yRef
0	0	0	0	0	0
1	1	0	0	0	0
2	0	1	0	1	1
3	1	1	0	1	1
4	0	0	1	0	0
5	1	0	1	1	1
6	0	1	1	0	0
7	1	1	1	1	1
8	1	0	1	1	1
9	0	1	1	0	0
10	1	0	1	1	1
11	1	1	0	1	1
13	1	0	1	1	1
14	0	0	0	0	0
15	1	1	1	1	1
16	1	1	0	1	1
17	0	1	1	0	0


```
In [14]: Test=(MUX2_1_ComboData['y']==MUX2_1_ComboData['yRef']).all()
print(f'Module `MUX2_1_Combo` works as expected: {Test}')
```

Module `MUX2_1_Combo` works as expected: True

▼ 4.4 Verilog Conversion

```
In [15]: DUT.convert()
VerilogTextReader('MUX2_1_Combo');
```

Verilog modular from MUX2_1_Combo.v

```
// File: MUX2_1_Combo.v
// Generated by MyHDL 0.10
// Date: Sat Sep 22 02:11:10 2018
```

```
`timescale 1ns/10ps
```

```
module MUX2_1_Combo (
    x0,
    x1,
    s,
    y
);
// 2:1 Multiplexer written in full combo
// Input:
//    x0(bool): input channel 0
//    x1(bool): input channel 1
//    s(bool): channel selection input
```

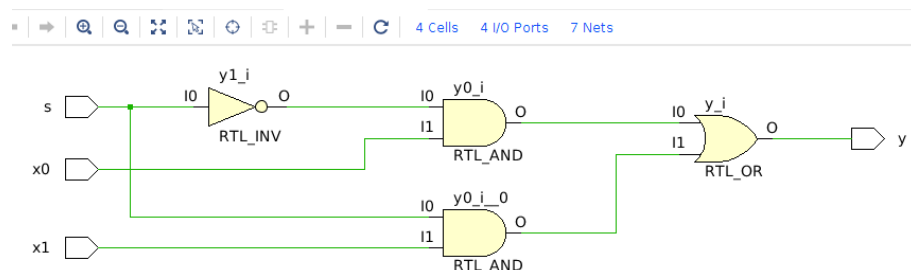


Figure 2: MUX2_1_Combo RTL schematic; Xilinx Vivado 2017.4

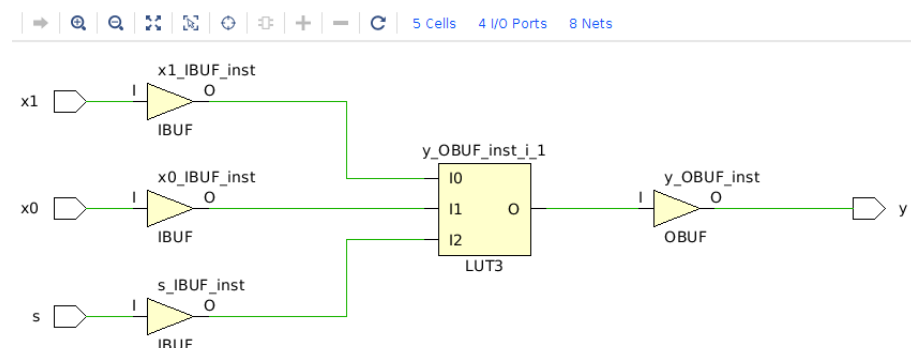


Figure 3: MUX2_1_Combo Synthesized Schematic; Xilinx Vivado 2017.4

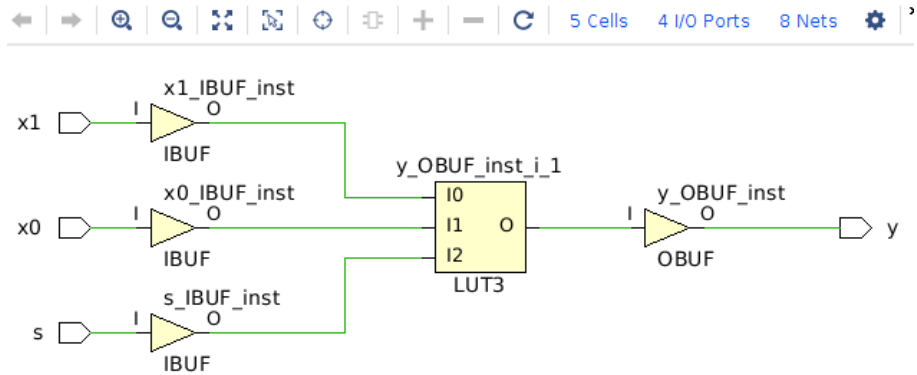


Figure 4: MUX2_1_Combio Implemented Schematic; Xilinx Vivado 2017.4

▼ 4.5 myHDL to Verilog Testbench

```
In [16]: ▼ #create BitVectors for AndGate_TBV
x0TVs=intbv(int(''.join(x0TVs.astype(str)), 2))[TestLen:]
x1TVs=intbv(int(''.join(x1TVs.astype(str)), 2))[TestLen:]
sTVs=intbv(int(''.join(sTVs.astype(str)), 2))[TestLen:]

x0TVs, bin(x0TVs), x1TVs, bin(x1TVs), sTVs, bin(sTVs)
```

```
Out[16]: (intbv(52583),
          '1100110101100111',
          intbv(87798),
          '10101011011110110',
          intbv(16277),
          '11111110010101')
```

```
In [17]: @block
def MUX2_1_Combo_TBV():
    """
    myHDL -> Verilog testbench for module `MUX2_1_Combo`
    """
    x0=Signal(bool(0))
    x1=Signal(bool(0))
    s=Signal(bool(0))
    y=Signal(bool(0))

    @always_comb
    def print_data():
        print(x0, x1, s, y)

    #Test Signal Bit Vectors
    x0TV=Signal(x0TVs)
    x1TV=Signal(x1TVs)
    sTV=Signal(sTVs)

    DUT=MUX2_1_Combo(x0, x1, s, y)

    @instance
    def stimules():
        for i in range(TestLen):
            x0.next=int(x0TV[i])
            x1.next=int(x1TV[i])
            s.next=int(sTV[i])
            yield delay(1)

        raise StopSimulation()
    return instances()

TB=MUX2_1_Combo_TBV()
TB.convert(hdl="Verilog", initial_values=True)
VerilogTextReader('MUX2_1_Combo_TBV');
```

```
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
***Verilog modular from MUX2_1_Combo_TBV.v***
```

```
// File: MUX2_1_Combo_TBV.v
// Generated by MyHDL 0.10
// Date: Sat Sep 22 02:11:15 2018
```

```
`timescale 1ns/10ps
```

```
module MUX2_1_Combo_TBV (
```

```
);
```

```
// myHDL -> Verilog testbench for module `MUX2_1_Combo`
```

```
reg x0 = 0;
```

```

reg x1 = 0;
reg s = 0;
wire y;
wire [17:0] x0TV;
wire [17:0] x1TV;
wire [17:0] sTV;

assign x0TV = 18'd52583;
assign x1TV = 18'd87798;
assign sTV = 18'd16277;

always @(x0, x1, s, y) begin: MUX2_1_COMBO_TBV_PRINT_DATA
    $write("%h", x0);
    $write(" ");
    $write("%h", x1);
    $write(" ");
    $write("%h", s);
    $write(" ");
    $write("%h", y);
    $write("\n");
end

assign y = (((!s) && x0) | (s && x1));

initial begin: MUX2_1_COMBO_TBV_STIMULES
    integer i;
    for (i=0; i<18; i=i+1) begin
        x0 <= x0TV[i];
        x1 <= x1TV[i];
        s <= sTV[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
/home/iridium/anaconda3/lib/python3.6/site-packages/myhdl/conversion/_t
oVerilog.py:349: ToVerilogWarning: Signal is not driven: sTV
    category=ToVerilogWarning

```

▼ 4.6 PYNQ-Z1 Deployment

▼ 4.6.1 Board Circuit

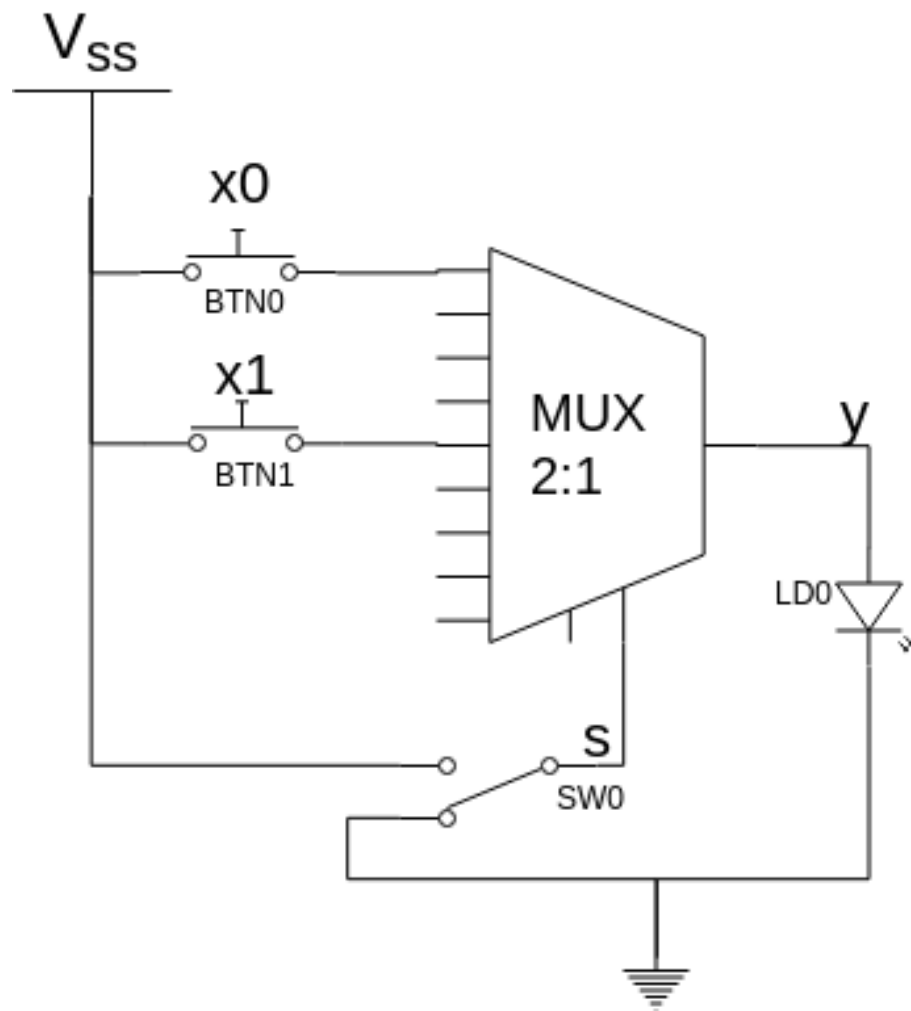


Figure 5: 2:1 MUX PYNQ-Z1 (Non SoC) conceptualized circuit

▼ 4.6.2 Board Constraint

```
In [18]: ConstraintXDCTextReader('MUX2_1');

***Constraint file from MUX2_1.xdc***

## PYNQ-Z1 Constraint File for MUX2_1_Combo
## Based on https://github.com/Xilinx/PYNQ/blob/master/boards/Pynq-Z1/base/vivado/constraints/base.xdc (https://github.com/Xilinx/PYNQ/blob/master/boards/Pynq-Z1/base/vivado/constraints/base.xdc)

## Switches
set_property -dict {PACKAGE_PIN M20 IOSTANDARD LVCMOS33} [get_ports {s}]; ##SW0

## Buttons
set_property -dict {PACKAGE_PIN D19 IOSTANDARD LVCMOS33} [get_ports {x0}]; ## BT0
set_property -dict {PACKAGE_PIN D20 IOSTANDARD LVCMOS33} [get_ports {x1}]; ##BT1

## LEDs
set_property -dict {PACKAGE_PIN R14 IOSTANDARD LVCMOS33} [get_ports {y}]; ## Led 0
```

▼ 4.6.3 Video of Deployment

MUX2_1_Combo myHDL PYNQ-Z1 ([YouTube \(https://www.youtube.com/watch?v=UkXbnFdF010\)](https://www.youtube.com/watch?v=UkXbnFdF010))

▼ 5 4 Channel Input : 1 Channel Output multiplexer in Gate Level Logic

▼ 5.1 Sympy Expression

```
In [19]: x0, x1, x2, x3, s0, s1, y=symbols('x0, x1, x2, x3, s0, s1, y')
y41Eq=Eq(y, (~s0&~s1&x0) | (s0&~s1&x1) | (~s0&s1&x2) | (s0&s1&x3))
y41Eq
```

```
Out[19]:  $y = (s_0 \wedge s_1 \wedge x_3) \vee (s_0 \wedge x_1 \wedge \neg s_1) \vee (s_1 \wedge x_2 \wedge \neg s_0) \vee (x_0 \wedge \neg s_0 \wedge \neg s_1)$ 
```

```
In [20]: TruthTabelGenrator(y41Eq)[[x3, x2, x1, x0, s1, s0, y]]
```

```
Out[20]:
```

	x3	x2	x1	x0	s1	s0	y
0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0
2	0	1	0	0	0	0	0
3	1	1	0	0	0	0	0
4	0	0	1	0	0	0	0
5	1	0	1	0	0	0	0
6	0	1	1	0	0	0	0
7	1	1	1	0	0	0	0
8	0	0	0	1	0	0	1
9	1	0	0	1	0	0	1
10	0	1	0	1	0	0	1

```
In [21]: y41EqN=lambdify([x0, x1, x2, x3, s0, s1], y41Eq.rhs, dummify=False)
SystematicVals=np.array(list(itertools.product([0,1], repeat=6)))
SystematicVals
y41EqN(*[SystematicVals[:, i] for i in range(6)] ).astype(int)
```

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

▼ 5.2 myHDL Module

```
In [22]: @block
▼ def MUX4_1_Combo(x0, x1, x2, x3, s0, s1, y):
    """
    4:1 Multiplexer written in full combo
    Input:
        x0(bool): input channel 0
        x1(bool): input channel 1
        x2(bool): input channel 2
        x3(bool): input channel 3
        s1(bool): channel selection input bit 1
        s0(bool): channel selection input bit 0
    Output:
        y(bool): ouput
    """

    @always_comb
    ▼ def logic():
        y.next= (not s0 and not s1 and x0) or (s0 and not s1 and x1)

    return instances()
```

▼ 5.3 myHDL Testing


```

In [23]: ▾ #generate systmatic and random test values
TestLen=5
SystmaticVals=list(itertools.product([0,1], repeat=6))

s0TVs=np.array([i[0] for i in SystmaticVals]).astype(int)
np.random.seed(15)
s0TVs=np.append(s0TVs, np.random.randint(0,2, TestLen)).astype(int)

s1TVs=np.array([i[1] 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)
s1TVs=np.append(s1TVs, np.random.randint(0,2, TestLen)).astype(int)

x0TVs=np.array([i[2] 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(17)
x0TVs=np.append(x0TVs, np.random.randint(0,2, TestLen)).astype(int)

x1TVs=np.array([i[3] 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(18)
x1TVs=np.append(x1TVs, np.random.randint(0,2, TestLen)).astype(int)

x2TVs=np.array([i[4] 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(19)
x2TVs=np.append(x2TVs, np.random.randint(0,2, TestLen)).astype(int)

x3TVs=np.array([i[5] 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(20)
x3TVs=np.append(x3TVs, np.random.randint(0,2, TestLen)).astype(int)

TestLen=len(x0TVs)
SystmaticVals, s0TVs, s1TVs, x3TVs, x2TVs, x1TVs, x0TVs, TestLen

```

```

Out[23]: ([ (0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 1),
(0, 0, 0, 0, 1, 0),
(0, 0, 0, 0, 1, 1),
(0, 0, 0, 1, 0, 0),
(0, 0, 0, 1, 0, 1),
(0, 0, 0, 1, 1, 0),
(0, 0, 0, 1, 1, 1),
(0, 0, 1, 0, 0, 0),
(0, 0, 1, 0, 0, 1),
(0, 0, 1, 0, 1, 0),
(0, 0, 1, 0, 1, 1),

```

```
(0, 0, 1, 1, 0, 0),
(0, 0, 1, 1, 0, 1),
(0, 0, 1, 1, 1, 0),
(0, 0, 1, 1, 1, 1),
(0, 1, 0, 0, 0, 0),
(0, 1, 0, 0, 0, 1),
(0, 1, 0, 0, 1, 0),
(0, 1, 0, 0, 1, 1)
```

In [24]:

```
Peeker.clear()
x0=Signal(bool(0)); Peeker(x0, 'x0')
x1=Signal(bool(0)); Peeker(x1, 'x1')
x2=Signal(bool(0)); Peeker(x2, 'x2')
x3=Signal(bool(0)); Peeker(x3, 'x3')

s0=Signal(bool(0)); Peeker(s0, 's0')
s1=Signal(bool(0)); Peeker(s1, 's1')
y=Signal(bool(0)); Peeker(y, 'y')

DUT=MUX4_1_Combo(x0, x1, x2, x3, s0, s1, y)

def MUX4_1_Combo_TB():
    """
    myHDL only testbench for module `MUX4_1_Combo`
    """

    @instance
    def stimules():
        for i in range(TestLen):
            x0.next=int(x0TVs[i])
            x1.next=int(x1TVs[i])
            x2.next=int(x2TVs[i])
            x3.next=int(x3TVs[i])
            s0.next=int(s0TVs[i])
            s1.next=int(s1TVs[i])

            yield delay(1)

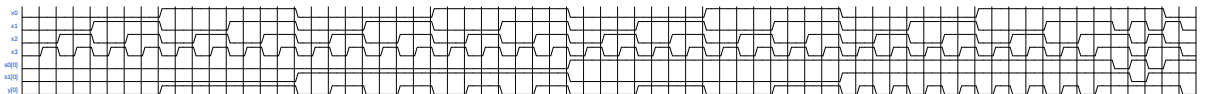
        raise StopSimulation()

    return instances()

sim=Simulation(DUT, MUX4_1_Combo_TB(), *Peeker.instances()).run()
```

In [25]:

```
Peeker.to_wavedrom()
```



```
In [26]: MUX4_1_ComboData=Peeker.to_dataframe()
MUX4_1_ComboData=MUX4_1_ComboData[['x3', 'x2', 'x1', 'x0', 's1', 's0',
MUX4_1_ComboData
```

Out[26]:

	x3	x2	x1	x0	s1	s0	y
0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0
2	0	1	0	0	0	0	0
3	1	1	0	0	0	0	0
4	0	0	1	0	0	0	0
5	1	0	1	0	0	0	0
6	0	1	1	0	0	0	0
7	1	1	1	0	0	0	0
8	0	0	0	1	0	0	1
9	1	0	0	1	0	0	1
10	0	1	0	1	0	0	1

```
In [27]: MUX4_1_ComboData['yRef']=MUX4_1_ComboData.apply(lambda row:y41EqN(row[
MUX4_1_ComboData
```

Out[27]:

	x3	x2	x1	x0	s1	s0	y	yRef
0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0
4	0	0	1	0	0	0	0	0
5	1	0	1	0	0	0	0	0
6	0	1	1	0	0	0	0	0
7	1	1	1	0	0	0	0	0
8	0	0	0	1	0	0	1	1
9	1	0	0	1	0	0	1	1
10	0	1	0	1	0	0	1	1

```
In [28]: Test=(MUX4_1_ComboData['y']==MUX4_1_ComboData['yRef']).all()
print(f'Module `MUX4_1_Combo` works as exspected: {Test}')
```

Module `MUX4_1_Combo` works as exspected: True

In [29]:

```
DUT.convert()
VerilogTextReader('MUX4_1_Combo');

***Verilog modular from MUX4_1_Combo.v***

// File: MUX4_1_Combo.v
// Generated by MyHDL 0.10
// Date: Sat Sep 22 02:11:48 2018

`timescale 1ns/10ps

module MUX4_1_Combo (
    x0,
    x1,
    x2,
    x3,
    s0,
    s1,
    y
);
// 4:1 Multiplexer written in full combo
// Input:
//     x0(bool): input channel 0
//     x1(bool): input channel 1
//     x2(bool): input channel 2
//     x3(bool): input channel 3
//     s1(bool): channel selection input bit 1
//     s0(bool): channel selection input bit 0
// Output:
//     y(bool): output

input x0;
input x1;
input x2;
input x3;
input s0;
input s1;
output y;
wire y;

assign y = (((!s0) && (!s1) && x0) || (s0 && (!s1) && x1) || ((!s0) &&
s1 && x2) || (s0 && s1 && x3));

endmodule
```

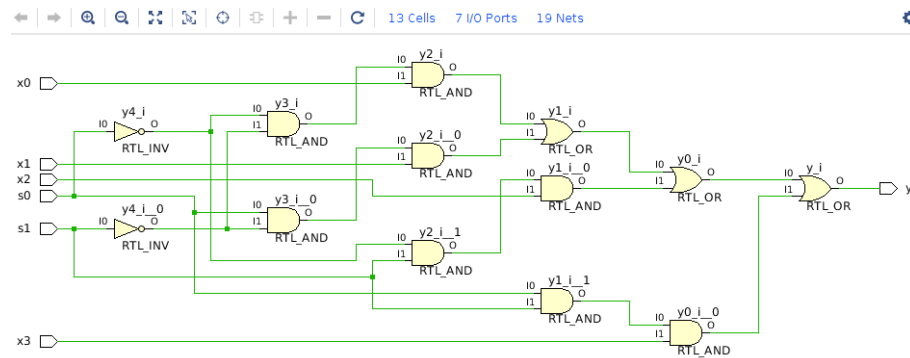


Figure 6: MUX4_1_Combo RTL schematic; Xilinx Vivado 2017.4

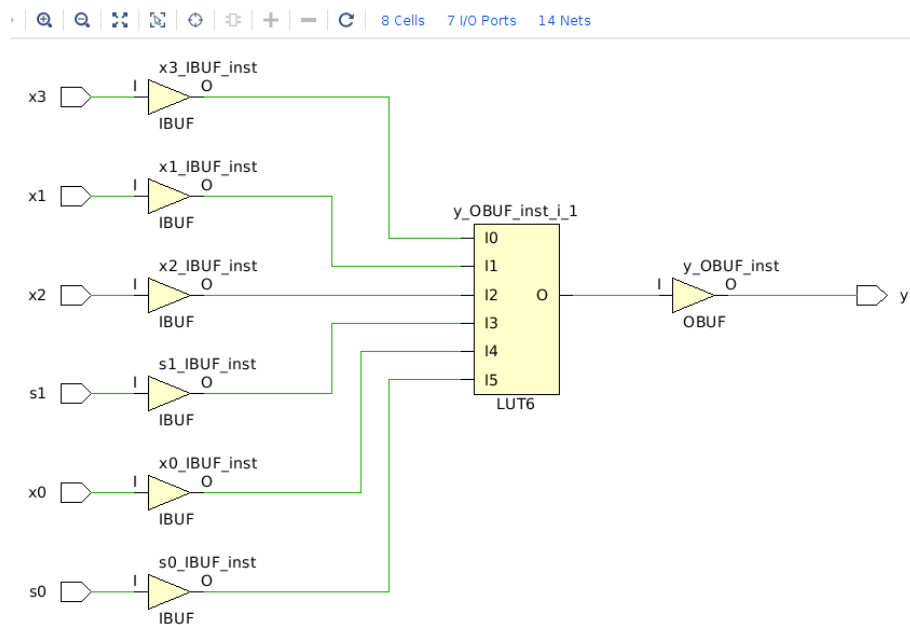


Figure 7: MUX4_1_Combo Synthesized Schematic; Xilinx Vivado 2017.4

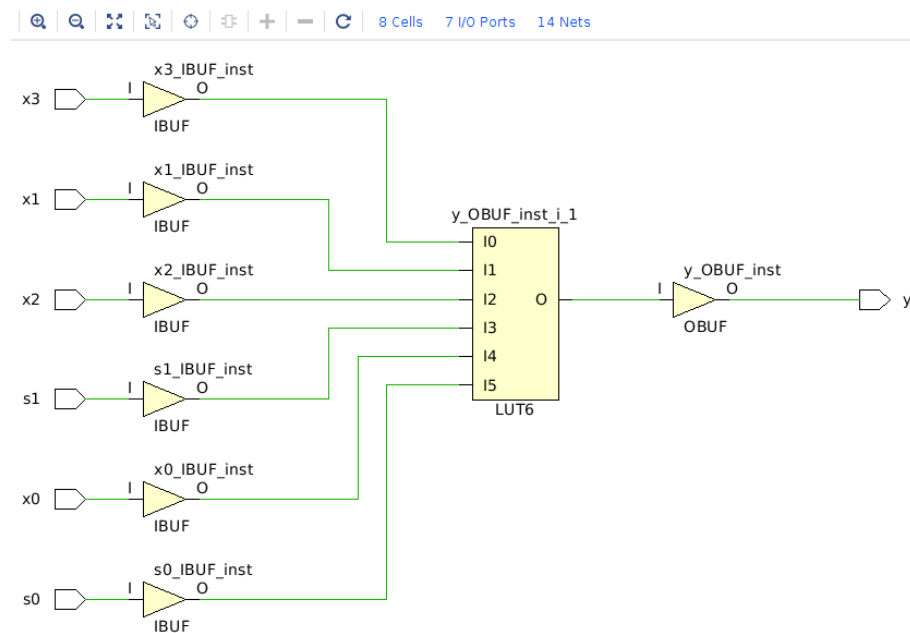


Figure 8: MUX4_1_Combo Implemented Schematic; Xilinx Vivado 2017.4

▼ 5.5 myHDL to Verilog Testbench

```
In [30]: #create BitVectors for MUX4_1_Combo_TBV
x0TVs=intbv(int(''.join(x0TVs.astype(str)), 2))[TestLen:]
x1TVs=intbv(int(''.join(x1TVs.astype(str)), 2))[TestLen:]
x2TVs=intbv(int(''.join(x2TVs.astype(str)), 2))[TestLen:]
x3TVs=intbv(int(''.join(x3TVs.astype(str)), 2))[TestLen:]


s0TVs=intbv(int(''.join(s0TVs.astype(str)), 2))[TestLen:]
s1TVs=intbv(int(''.join(s1TVs.astype(str)), 2))[TestLen:]


x0TVs, bin(x0TVs), x1TVs, bin(x1TVs), x2TVs, bin(x2TVs), x3TVs, bin(x3TVs)

Out[30]: (intbv(2296870857426870268),
'111111110000000011111111000000001111111100000000111111111100',
intbv(34723282962276803050),
'11110000111100001111000011110000111100001111000011110000111101010',
intbv(118059162071741130356),
'110011001100110011001100110011001100110011001100110011001100110100',
intbv(196765270119568550582),
'1010101010101010101010101010101010101010101010101010101010101011
0',
intbv(137438953451),
'1111111111111111111111111111111111111111111101011',
intbv(9007061817884663),
'111111111111111111000000000000000000000000011111111111111111111')

```

In [31]:

```
@block
def MUX4_1_Combo_TBV():
    """
    myHDL -> Verilog testbench for module `MUX4_1_Combo`
    """

    x0=Signal(bool(0))
    x1=Signal(bool(0))
    x2=Signal(bool(0))
    x3=Signal(bool(0))
    y=Signal(bool(0))
    s0=Signal(bool(0))
    s1=Signal(bool(0))

    @always_comb
    def print_data():
        print(x0, x1, x2, x3, s0, s1, y)

    #Test Signal Bit Vectors
    x0TV=Signal(x0TVs)
    x1TV=Signal(x1TVs)
    x2TV=Signal(x2TVs)
    x3TV=Signal(x3TVs)
    s0TV=Signal(s0TVs)
    s1TV=Signal(s1TVs)

    DUT=MUX4_1_Combo(x0, x1, x2, x3, s0, s1, y)

    @instance
    def stimulates():
        for i in range(TestLen):
            x0.next=int(x0TV[i])
            x1.next=int(x1TV[i])
            x2.next=int(x2TV[i])
            x3.next=int(x3TV[i])
            s0.next=int(s0TV[i])
            s1.next=int(s1TV[i])
            yield delay(1)

        raise StopSimulation()
    return instances()

TB=MUX4_1_Combo_TBV()
TB.convert(hdl="Verilog", initial_values=True)
VerilogTextReader('MUX4_1_Combo_TBV');
```

```
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
```

Typesetting math: 100%***Verilog modular from MUX4_1_Combo_TBV.v***

```
// File: MUX4_1_Combo_TBV.v
// Generated by MyHDL 0.10
// Date: Sat Sep 22 02:11:52 2018
```

```
`timescale 1ns/10ps
```

```
module MUX4_1_Combo_TBV (
```

```
);
```

```
// myHDL -> Verilog testbench for module `MUX4_1_Combo`
```

```
reg x0 = 0;
reg x1 = 0;
wire y;
reg x2 = 0;
reg x3 = 0;
reg s0 = 0;
reg s1 = 0;
wire [68:0] x0TV;
wire [68:0] x1TV;
wire [68:0] x2TV;
wire [68:0] x3TV;
wire [68:0] s0TV;
wire [68:0] s1TV;
```

```
assign x0TV = 69'd2296870857426870268;
assign x1TV = 69'd34723282962276803050;
assign x2TV = 69'd118059162071741130356;
assign x3TV = 69'd196765270119568550582;
assign s0TV = 69'd137438953451;
assign s1TV = 69'd9007061817884663;
```

```
always @(x1, x2, s1, y, s0, x0, x3) begin: MUX4_1_COMBO_TBV_PRINT_DATA
    $write("%h", x0);
    $write(" ");
    $write("%h", x1);
    $write(" ");
    $write("%h", x2);
    $write(" ");
    $write("%h", x3);
    $write(" ");
    $write("%h", s0);
    $write(" ");
    $write("%h", s1);
    $write(" ");
    $write("%h", y);
    $write("\n");
end
```

```
assign y = (((!s0) && (!s1) && x0) || (s0 && (!s1) && x1) || ((!s0) &&
s1 && x2) || (s0 && s1 && x3));
```



```

initial begin: MUX4_1_COMBO_TBV_STIMULES
    integer i;
    for (i=0; i<69; i=i+1) begin
        x0 <= x0TV[i];
        x1 <= x1TV[i];
        x2 <= x2TV[i];
        x3 <= x3TV[i];
        s0 <= s0TV[i];
        s1 <= s1TV[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
/home/iridium/anaconda3/lib/python3.6/site-packages/myhdl/conversion/_t
oVerilog.py:349: ToVerilogWarning: Signal is not driven: x2TV
    category=ToVerilogWarning
/home/iridium/anaconda3/lib/python3.6/site-packages/myhdl/conversion/_t
oVerilog.py:349: ToVerilogWarning: Signal is not driven: x3TV
    category=ToVerilogWarning
/home/iridium/anaconda3/lib/python3.6/site-packages/myhdl/conversion/_t
oVerilog.py:349: ToVerilogWarning: Signal is not driven: s0TV
    category=ToVerilogWarning
/home/iridium/anaconda3/lib/python3.6/site-packages/myhdl/conversion/_t
oVerilog.py:349: ToVerilogWarning: Signal is not driven: s1TV
    category=ToVerilogWarning

```

▼ 5.6 PYNQ-Z1 Deployment

▼ 5.6.1 Board Circuit

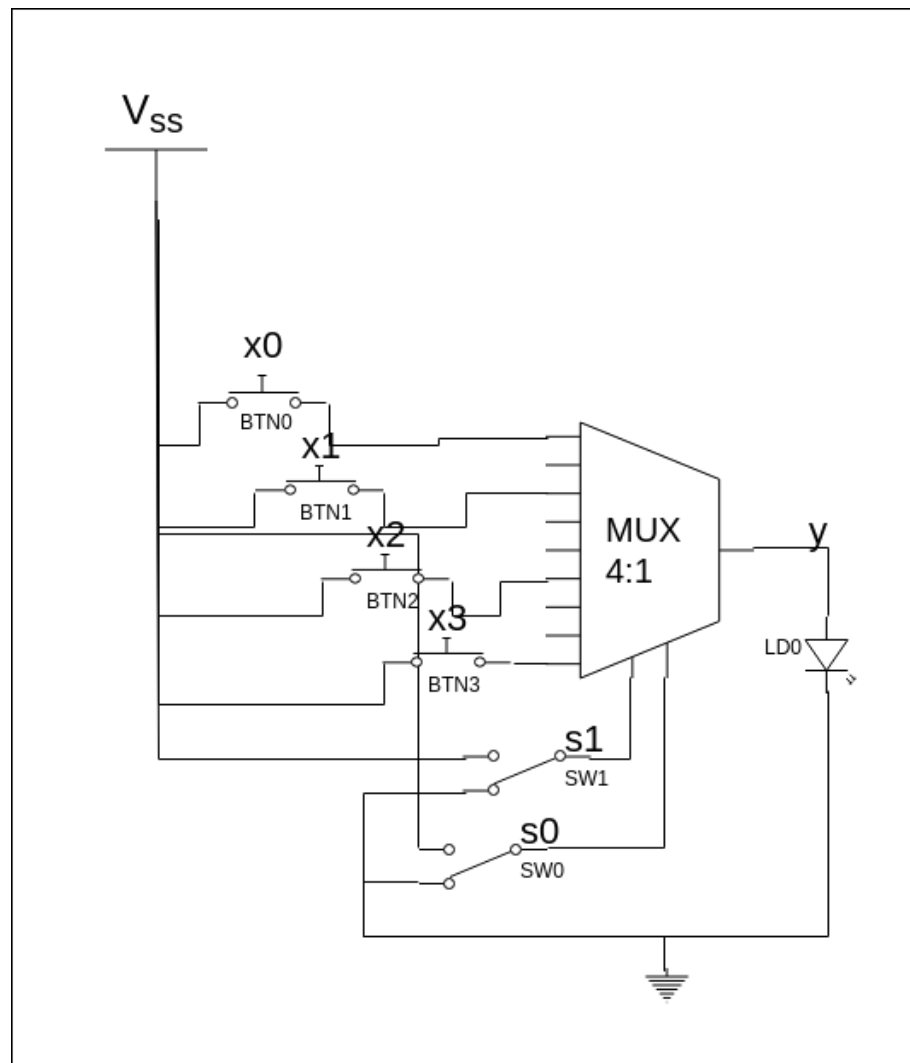


Figure 9: 4:1 MUX PYNQ-Z1 (Non SoC) conceptualized circuit

▼ 5.6.2 Board Constraint

In [32]: `ConstraintXDCTextReader('MUX4_1');`

```
***Constraint file from MUX4_1.xdc***
```

```
## PYNQ-Z1 Constraint File for MUX4_1_*  
## Based on https://github.com/Xilinx/PYNQ/blob/master/boards/Pynq-Z1/base/vivado/constraints/base.xdc (https://github.com/Xilinx/PYNQ/blob/master/boards/Pynq-Z1/base/vivado/constraints/base.xdc)
```

```
## Switches  
set_property -dict {PACKAGE_PIN M20 IOSTANDARD LVCMOS33} [get_ports {s0}]; ## SW0  
set_property -dict {PACKAGE_PIN M19 IOSTANDARD LVCMOS33} [get_ports {s1}]; ## SW1
```

```
## Buttons  
set_property -dict {PACKAGE_PIN D19 IOSTANDARD LVCMOS33} [get_ports {x0}]; ## BT0  
set_property -dict {PACKAGE_PIN D20 IOSTANDARD LVCMOS33} [get_ports {x1}]; ## BT1  
set_property -dict {PACKAGE_PIN L20 IOSTANDARD LVCMOS33} [get_ports {x2}]; ## BT2  
set_property -dict {PACKAGE_PIN L19 IOSTANDARD LVCMOS33} [get_ports {x3}]; ## BT3
```

```
## LEDs  
set_property -dict {PACKAGE_PIN R14 IOSTANDARD LVCMOS33} [get_ports {y}]; ## Led 0
```

▼ 5.6.3 Video of Deployment

MUX4_1_MS myHDL PYNQ-Z1 ([YouTube \(https://www.youtube.com/watch?v=ZdwGjRM9Wfk\)](https://www.youtube.com/watch?v=ZdwGjRM9Wfk))

▼ 6 Shannon's Expansion Formula & Stacking of MUXs

[Claude Shannon \(https://en.wikipedia.org/wiki/Claude_Shannon\)](https://en.wikipedia.org/wiki/Claude_Shannon), of the famed Shannon-Nyquist theorem, discovered that any boolean expression $F(x_0, x_1, \dots, x_n)$ can be decomposed in a manner akin to polynomials of perfect squares via

$$F(x_0, x_1, \dots, x_n) = x_0 \cdot F(x_0 = 1, x_1, \dots, x_n) + \overline{x_0} \cdot F(x_0 = 0, x_1, \dots, x_n)$$

known as the Sum of Products (SOP) form since when the expansion is completed for all x_n the result is that

$$F(x_0, x_1, \dots, x_n) = \sum_{i=0}^{2^n-1} (m_i \cdot F(m_i))$$

aka the Sum of all Minterms (m_i) belonging to the original boolean expression F factored down to the i th of n variables belonging to F and product (&) of F evaluated with the respective minterm as the argument

The Dual to the SOP form of Shannon's expansion formula is the Product of Sum (POS) form

$$F(x_0, x_1, \dots, x_n) = (x_0 + F(x_0 = 1, x_1, \dots, x_n)) \cdot (\overline{x_0} + F(x_0 = 0, x_1, \dots, x_n))$$

thus

$$F(x_0, x_1, \dots, x_n) = \prod_{i=0}^{2^n-1} (M_i + F(M_i))$$

with M_i being the i th Maxterm

it is for this reason that Shannon's Expansion Formula is known is further linked to the fundamental theorem of algebra that it is called the "fundamental theorem of Boolean algebra"

So why then is Shannon's decomposition formula discussed in terms of Multiplexers. Because the general expression for a $2^n : 1$ multiplexer is

$$y_{\text{MUX}} = \sum_{i=0}^{2^n-1} m_i \cdot x_n$$

where then n is the required number of control inputs (referred to in this tutorial as s_i). Which is the same as the SOP form of Shannon's Formula for a boolean expression that has been fully decomposed (Factored). And further, if the boolean expression has not been fully factored we can replace $n - 1$ parts of the partially factored expression with multiplexers. This then gives way to what is called "Multiplexer Stacking" in order to implement large boolean expressions and or large multiplexers

▼ 7 4 Channel Input: 1 Channel Output multiplexer via MUX Stacking

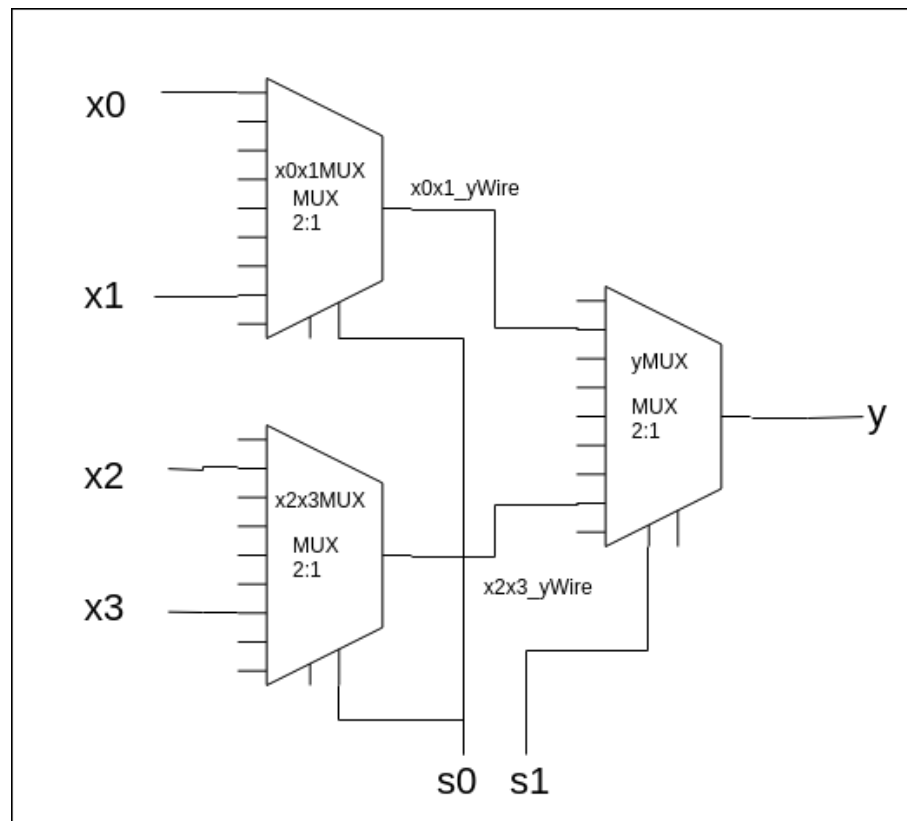


Figure 10: 4:1 MUX via MUX stacking 2:1MUXs

▼ 7.1 myHDL Module

```

In [33]: @block
▼ def MUX4_1_MS(x0, x1, x2, x3, s0, s1, y):
    """
    4:1 Multiplexer via 2:1 MUX stacking
    Input:
        x0(bool): input channel 0
        x1(bool): input channel 1
        x2(bool): input channel 2
        x3(bool): input channel 3
        s1(bool): channel selection input bit 1
        s0(bool): channel selection input bit 0
    Output:
        y(bool): ouput
    """
    #create ouput from x0x1 input MUX to y ouput MUX
    x0x1_yWire=Signal(bool(0))
    #create instance of 2:1 mux and wire in inputs
    #a, b, s0 and wire to ouput mux
    x0x1MUX=MUX2_1_Combo(x0, x1, s0, x0x1_yWire)

    #create ouput from x2x3 input MUX to y ouput MUX
    x2x3_yWire=Signal(bool(0))
    #create instance of 2:1 mux and wire in inputs
    #c, d, s0 and wire to ouput mux
    x2x3MUX=MUX2_1_Combo(x2, x3, s0, x2x3_yWire)

    #create ouput MUX and wire to internal wires,
    #s1 and ouput y
    yMUX=MUX2_1_Combo(x0x1_yWire, x2x3_yWire, s1, y)

    return instances()

```

▼ 7.2 myHDL Testing

```

In [34]: ▾ #generate systmatic and random test values
TestLen=5
SystmaticVals=list(itertools.product([0,1], repeat=6))

s0TVs=np.array([i[0] for i in SystmaticVals]).astype(int)
np.random.seed(15)
s0TVs=np.append(s0TVs, np.random.randint(0,2, TestLen)).astype(int)

s1TVs=np.array([i[1] 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)
s1TVs=np.append(s1TVs, np.random.randint(0,2, TestLen)).astype(int)

x0TVs=np.array([i[2] 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(17)
x0TVs=np.append(x0TVs, np.random.randint(0,2, TestLen)).astype(int)

x1TVs=np.array([i[3] 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(18)
x1TVs=np.append(x1TVs, np.random.randint(0,2, TestLen)).astype(int)

x2TVs=np.array([i[4] 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(19)
x2TVs=np.append(x2TVs, np.random.randint(0,2, TestLen)).astype(int)

x3TVs=np.array([i[5] 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(20)
x3TVs=np.append(x3TVs, np.random.randint(0,2, TestLen)).astype(int)

TestLen=len(x0TVs)
SystmaticVals, s0TVs, s1TVs, x3TVs, x2TVs, x1TVs, x0TVs, TestLen

```

```

Out[34]: ([ (0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 1),
(0, 0, 0, 0, 1, 0),
(0, 0, 0, 0, 1, 1),
(0, 0, 0, 1, 0, 0),
(0, 0, 0, 1, 0, 1),
(0, 0, 0, 1, 1, 0),
(0, 0, 0, 1, 1, 1),
(0, 0, 1, 0, 0, 0),
(0, 0, 1, 0, 0, 1),
(0, 0, 1, 0, 1, 0),
(0, 0, 1, 0, 1, 1),

```

```
(0, 0, 1, 1, 0, 0),
(0, 0, 1, 1, 0, 1),
(0, 0, 1, 1, 1, 0),
(0, 0, 1, 1, 1, 1),
(0, 1, 0, 0, 0, 0),
(0, 1, 0, 0, 0, 1),
(0, 1, 0, 0, 1, 0),
(0, 1, 0, 0, 1, 1)
```

In [35]:

```
Peeker.clear()
x0=Signal(bool(0)); Peeker(x0, 'x0')
x1=Signal(bool(0)); Peeker(x1, 'x1')
x2=Signal(bool(0)); Peeker(x2, 'x2')
x3=Signal(bool(0)); Peeker(x3, 'x3')

s0=Signal(bool(0)); Peeker(s0, 's0')
s1=Signal(bool(0)); Peeker(s1, 's1')
y=Signal(bool(0)); Peeker(y, 'y')

DUT=MUX4_1_MS(x0, x1, x2, x3, s0, s1, y)

▼ def MUX4_1_MS_TB():
    """
    myHDL only testbench for module `MUX4_1_MS`
    """

    @instance
    ▼ def stimules():
        ▼ for i in range(TestLen):
            x0.next=int(x0TVs[i])
            x1.next=int(x1TVs[i])
            x2.next=int(x2TVs[i])
            x3.next=int(x3TVs[i])
            s0.next=int(s0TVs[i])
            s1.next=int(s1TVs[i])

            yield delay(1)

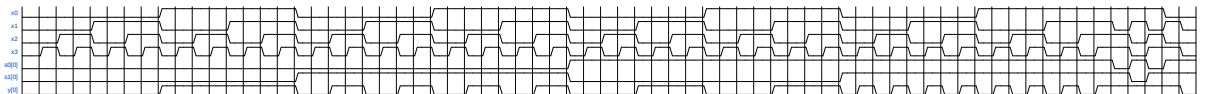
            raise StopSimulation()

        return instances()

sim=Simulation(DUT, MUX4_1_MS_TB(), *Peeker.instances()).run()
```

In [36]:

```
Peeker.to_wavedrom()
```




```
In [37]: MUX4_1_MSData=Peeker.to_dataframe()
MUX4_1_MSData=MUX4_1_MSData[['x3', 'x2', 'x1', 'x0', 's1', 's0', 'y']]
MUX4_1_MSData
```

Out[37]:

	x3	x2	x1	x0	s1	s0	y
0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0
2	0	1	0	0	0	0	0
3	1	1	0	0	0	0	0
4	0	0	1	0	0	0	0
5	1	0	1	0	0	0	0
6	0	1	1	0	0	0	0
7	1	1	1	0	0	0	0
8	0	0	0	1	0	0	1
9	1	0	0	1	0	0	1
10	0	1	0	1	0	0	1

```
In [38]: Test=MUX4_1_ComboData[['x3', 'x2', 'x1', 'x0', 's1', 's0', 'y']]==MUX4_
Test=Test.all().all()
print(f'Module `MUX4_1_MS` works as expected: {Test}')
```

Module `MUX4_1_MS` works as expected: True

▼ 7.3 Verilog Conversion

```
In [39]: DUT.convert()  
VerilogTextReader('MUX4_1_MS');
```

```
***Verilog modular from MUX4_1_MS.v***
```

```
// File: MUX4_1_MS.v  
// Generated by MyHDL 0.10  
// Date: Sat Sep 22 02:12:22 2018
```

```
`timescale 1ns/10ps
```

```
module MUX4_1_MS (  
    x0,  
    x1,  
    x2,  
    x3,  
    s0,  
    s1,  
    y  
);  
// 4:1 Multiplexer via 2:1 MUX stacking  
// Input:  
//    x0(bool): input channel 0  
//    x1(bool): input channel 1  
//    x2(bool): input channel 2  
//    x3(bool): input channel 3  
//    s1(bool): channel selection input bit 1  
//    s0(bool): channel selection input bit 0  
// Output:  
//    y(bool): output  
  
input x0;  
input x1;  
input x2;  
input x3;  
input s0;  
input s1;  
output y;  
wire y;  
  
wire x0x1_yWire;  
wire MUX2_1_Comb0_0_y;  
  
assign x0x1_yWire = (((!s0) && x0) | (s0 && x1));  
  
assign MUX2_1_Comb0_0_y = (((!s0) && x2) | (s0 && x3));  
  
assign y = (((!s1) && x0x1_yWire) | (s1 && MUX2_1_Comb0_0_y));
```

endmodule

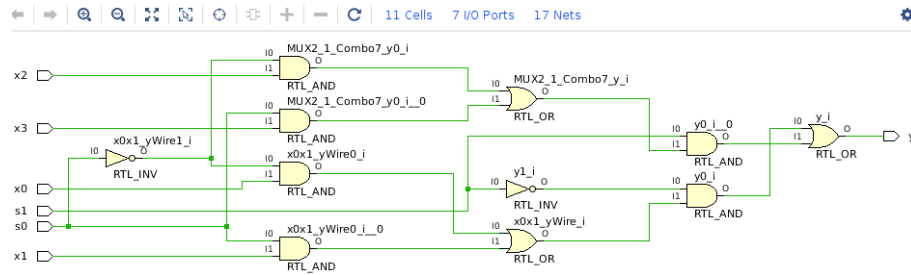


Figure 11: MUX4_1_MS RTL schematic; Xilinx Vivado 2017.4

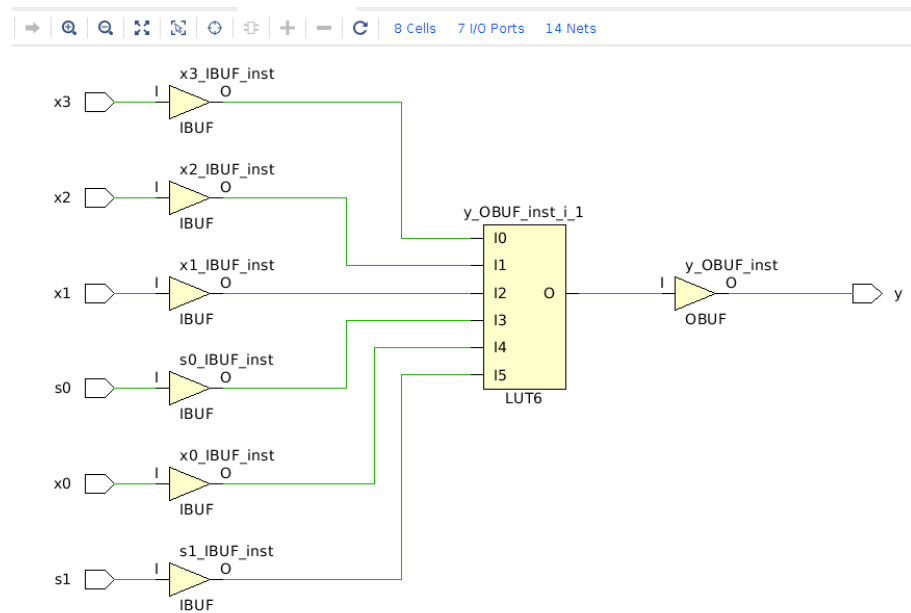


Figure 12: MUX4_1_MS Synthesized Schematic; Xilinx Vivado 2017.4

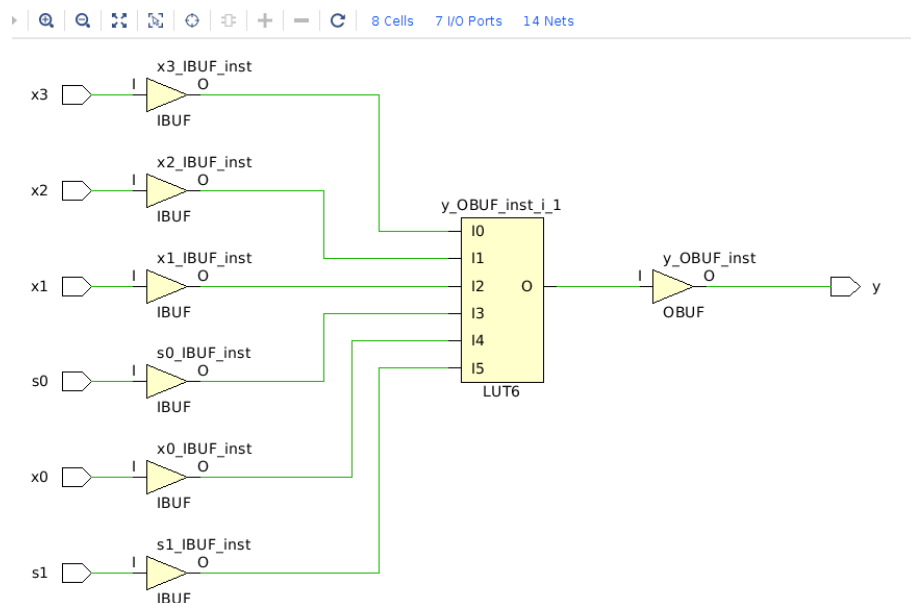


Figure 13: MUX4_1_MS Implemented Schematic; Xilinx Vivado 2017.4

7.4 myHDL to Verilog Testbench

```
In [40]: #create BitVectors for MUX4_1_Combo_TBV
x0TVs=intbv(int(''.join(x0TVs.astype(str)), 2))[TestLen:]
x1TVs=intbv(int(''.join(x1TVs.astype(str)), 2))[TestLen:]
x2TVs=intbv(int(''.join(x2TVs.astype(str)), 2))[TestLen:]
x3TVs=intbv(int(''.join(x3TVs.astype(str)), 2))[TestLen:]

s0TVs=intbv(int(''.join(s0TVs.astype(str)), 2))[TestLen:]
s1TVs=intbv(int(''.join(s1TVs.astype(str)), 2))[TestLen:]

x0TVs, bin(x0TVs), x1TVs, bin(x1TVs), x2TVs, bin(x2TVs), x3TVs, bin(x3TVs)
```

[illegible]

```

In [41]: @block
def MUX4_1_MS_TBV():
    """
    myHDL -> Verilog testbench for module `MUX4_1_MS`
    """

    x0=Signal(bool(0))
    x1=Signal(bool(0))
    x2=Signal(bool(0))
    x3=Signal(bool(0))
    y=Signal(bool(0))
    s0=Signal(bool(0))
    s1=Signal(bool(0))

    @always_comb
    def print_data():
        print(x0, x1, x2, x3, s0, s1, y)

    #Test Signal Bit Vectors
    x0TV=Signal(x0TVs)
    x1TV=Signal(x1TVs)
    x2TV=Signal(x2TVs)
    x3TV=Signal(x3TVs)
    s0TV=Signal(s0TVs)
    s1TV=Signal(s1TVs)

    DUT=MUX4_1_MS(x0, x1, x2, x3, s0, s1, y)

    @instance
    def stimules():
        for i in range(TestLen):
            x0.next=int(x0TV[i])
            x1.next=int(x1TV[i])
            x2.next=int(x2TV[i])
            x3.next=int(x3TV[i])
            s0.next=int(s0TV[i])
            s1.next=int(s1TV[i])
            yield delay(1)

        raise StopSimulation()
    return instances()

TB=MUX4_1_MS_TBV()
TB.convert(hdl="Verilog", initial_values=True)
VerilogTextReader('MUX4_1_MS_TBV');

```

```

<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>

```

```
// File: MUX4_1_MS_TBV.v
// Generated by MyHDL 0.10
// Date: Sat Sep 22 02:12:27 2018
```

```
`timescale 1ns/10ps
```

```
module MUX4_1_MS_TBV (
```

```
);
```

```
// MUX4_1_MS_TBV is a 4 Channel Input : 1 Channel Output multiplexer in Gate Level Logic
```

▼ 7.5 PYNQ-Z1 Deployment

▼ 7.5.1 Board Circuit

See Board Circuit for "4 Channel Input : 1 Channel Output multiplexer in Gate Level Logic"

▼ 7.5.2 Board Constraint

uses same 'MUX4_1.xdc' as "4 Channel Input : 1 Channel Output multiplexer in Gate Level Logic"

▼ 7.5.3 Video of Deployment

MUX4_1_MS myHDL PYNQ-Z1 ([YouTube \(https://www.youtube.com/watch?v=uO7VZ8ow_Yg\)](https://www.youtube.com/watch?v=uO7VZ8ow_Yg))

▼ 8 Introduction to HDL Behavioral Modeling

HDL behavioral modeling is a "High" level, though not at the HLS level, HDL syntax where the intended hardware element is modeled via its intended abstract algorithm behavior. Thus the common computer science (and mathematician) tool of abstraction is borrowed and incorporated into the HDL syntax. The abstraction that follows has, like all things, its pros and cons.

As a pro, this means that the Hard Ware Designer is no longer consumed by the manuchia of implementing boolean algebra for every device and can instead focus on implementing the intended algorithm in hardware. And it is thanks to this blending of Software and Hardware that the design of digital devices has grown as prolific as it has. However, there is quite a cache for using behavioral modeling. First off HDL now absolutely requires synthesis tools that can map the behavioral statements to hardware. And even when the behavioral logic is mapped at least to the RTL level there is no escaping two points. 1. At the end of the day, the RTL will be implemented via Gate level devices in some form or another. 2. the way the synthesis tool has mapped the abstract behavioral to RTL may not be physical implementable especially in ASIC implementations.

For these reasons it as Hardware Developers using Behavioral HDL we have to be able to still be able to implement the smallest indivisible units of our HDL at the gate level. Must know what physical limits our target architecture (FPGA, ASIC, etc) has and keep within these limits when writing our HDL code. And lastly, we can not grow lazy in writing behavioral HDL, but must always see at least down to the major RTL elements that our behavioral statements are embodying.

▼ 9 2:1 MUX via Behavioral IF

▼ 9.1 myHDL Module

```
In [42]: @block
▼ def MUX2_1_B(x0, x1, s, y):
    """
    2:1 Multiplexer written via behavioral if
    Input:
    x0(bool): input channel 0
    x1(bool): input channel 1
    s(bool): channel selection input
    Output:
    y(bool): ouput
    """

    @always_comb
    ▼ def logic():
    ▼     if s:
    ▼         y.next=x1
    ▼     else:
    ▼         y.next=x0

    return instances()
```

▼ 9.2 myHDL Testing

```

In [43]: ▾ #generate systmatic and random test values
TestLen=10
SystmaticVals=list(itertools.product([0,1], repeat=3))

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[2] for i in SystmaticVals]).astype(int)
#the random genrator must have a differint seed between 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)

sTVs=np.array([i[0] for i in SystmaticVals]).astype(int)
#the random genrator must have a differint seed between each generation
#call in order to produce differint values for each call
np.random.seed(17)
sTVs=np.append(sTVs, np.random.randint(0,2, TestLen)).astype(int)

TestLen=len(x0TVs)
x0TVs, x1TVs, sTVs, TestLen

```

```

Out[43]: (array([0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1]),
array([0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0]),
array([0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1]),
18)

```



```

In [44]: Peeker.clear()
x0=Signal(bool(0)); Peeker(x0, 'x0')
x1=Signal(bool(0)); Peeker(x1, 'x1')
s=Signal(bool(0)); Peeker(s, 's')
y=Signal(bool(0)); Peeker(y, 'y')

DUT=MUX2_1_B(x0, x1, s, y)

▼ def MUX2_1_B_TB():
    """
    myHDL only testbench for module `MUX2_1_B`
    """

    @instance
    ▼ def stimules():
        ▼ for i in range(TestLen):
            x0.next=int(x0TVs[i])
            x1.next=int(x1TVs[i])
            s.next=int(sTVs[i])

            yield delay(1)

            raise StopSimulation()

        return instances()

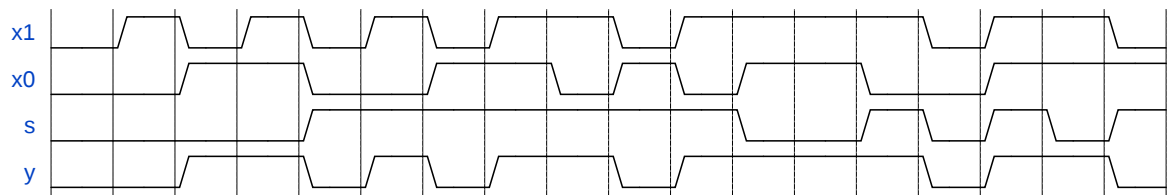
sim=Simulation(DUT, MUX2_1_B_TB(), *Peeker.instances()).run()

```

```

In [46]: Peeker.to_wavedrom('x1', 'x0', 's', 'y')

```



```
In [47]: MUX2_1_BData=Peeker.to_dataframe()
MUX2_1_BData=MUX2_1_BData[['x1', 'x0', 's', 'y']]
MUX2_1_BData
```

Out[47]:

	x1	x0	s	y
0	0	0	0	0
1	1	0	0	0
2	0	1	0	1
3	1	1	0	1
4	0	0	1	0
5	1	0	1	1
6	0	1	1	0
7	1	1	1	1
8	1	0	1	1
9	0	1	1	0
10	1	0	1	1
11	1	1	0	1
13	1	0	1	1
14	0	0	0	0
15	1	1	1	1
16	1	1	0	1
17	0	1	1	0

```
In [48]: Test=MUX2_1_ComboData[['x1', 'x0', 's', 'y']]==MUX2_1_BData
Test=Test.all().all()
print(f'`MUX2_1_B` Behavioral is Equivlint to `MUX2_1_Combo`: {Test}')
`MUX2_1_B` Behavioral is Equivlint to `MUX2_1_Combo`: True
```

▼ 9.3 Verilog Conversion

```
In [49]: DUT.convert()  
VerilogTextReader('MUX2_1_B');
```

```
***Verilog modular from MUX2_1_B.v***  
  
// File: MUX2_1_B.v  
// Generated by MyHDL 0.10  
// Date: Sat Sep 22 02:12:58 2018  
  
`timescale 1ns/10ps  
  
module MUX2_1_B (  
    x0,  
    x1,  
    s,  
    y  
);  
// 2:1 Multiplexer written via behavioral if  
// Input:  
//    x0(bool): input channel 0  
//    x1(bool): input channel 1  
//    s(bool): channel selection input  
// Output:  
//    y(bool): output  
  
input x0;  
input x1;  
input s;  
output y;  
reg y;  
  
always @(s, x1, x0) begin: MUX2_1_B_LOGIC  
    if (s) begin  
        y = x1;  
    end  
    else begin  
        y = x0;  
    end  
end  
  
endmodule
```

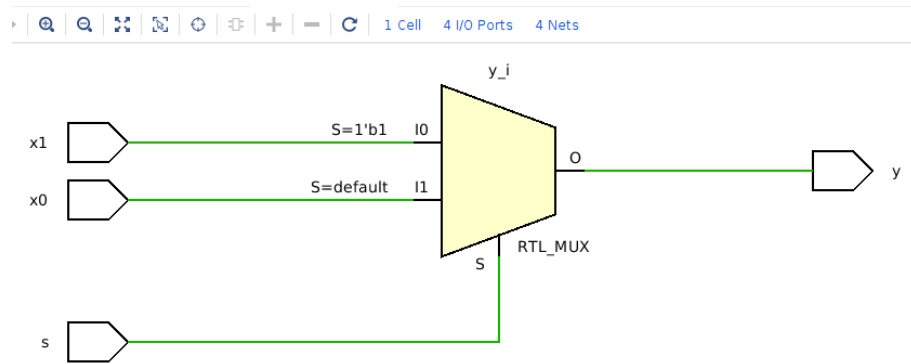


Figure 14: MUX2_1_B RTL schematic; Xilinx Vivado 2017.4

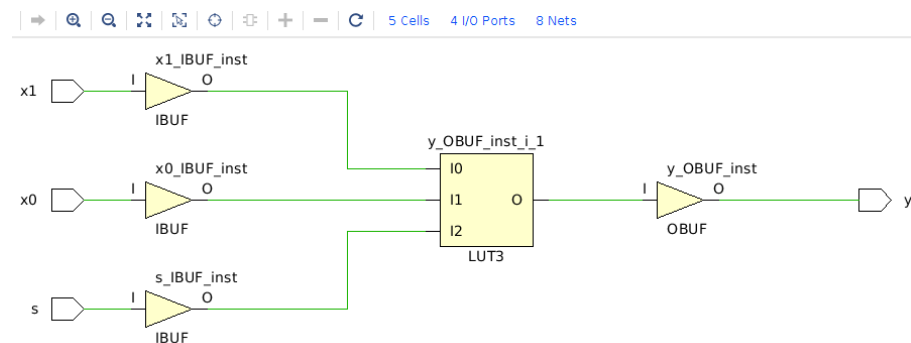


Figure 15: MUX2_1_B Synthesized Schematic; Xilinx Vivado 2017.4

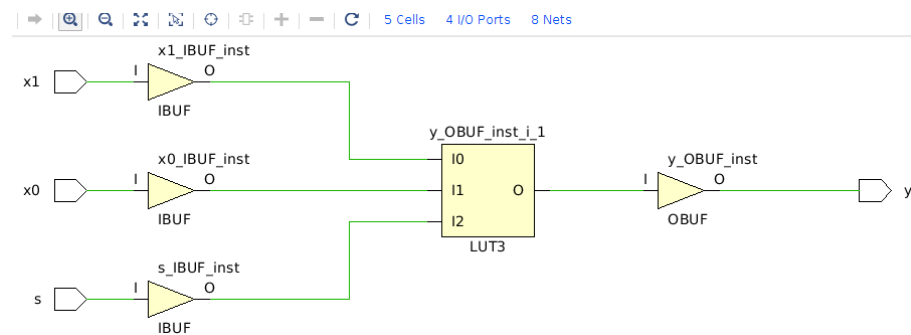


Figure 16: MUX2_1_B Implemented Schematic; Xilinx Vivado 2017.4

▼ 9.4 myHDL to Verilog Testbench

```
In [50]: #create BitVectors for AndGate_TBV  
x0TVs=intbv(int(''.join(x0TVs.astype(str)), 2))[TestLen:]  
x1TVs=intbv(int(''.join(x1TVs.astype(str)), 2))[TestLen:]  
sTVs=intbv(int(''.join(sTVs.astype(str)), 2))[TestLen:]  
  
x0TVs, bin(x0TVs), x1TVs, bin(x1TVs), sTVs, bin(sTVs)
```

```
Out[50]: (intbv(52583),  
          '1100110101100111',  
          intbv(87798),  
          '10101011011110110',  
          intbv(16277),  
          '11111110010101')
```

```
In [51]: @block
def MUX2_1_B_TBV():
    """
    myHDL -> Verilog testbench for module `MUX2_1_B`
    """
    x0=Signal(bool(0))
    x1=Signal(bool(0))
    s=Signal(bool(0))
    y=Signal(bool(0))

    @always_comb
    def print_data():
        print(x0, x1, s, y)

    #Test Signal Bit Vectors
    x0TV=Signal(x0TVs)
    x1TV=Signal(x1TVs)
    sTV=Signal(sTVs)

    DUT=MUX2_1_B(x0, x1, s, y)

    @instance
    def stimules():
        for i in range(TestLen):
            x0.next=int(x0TV[i])
            x1.next=int(x1TV[i])
            s.next=int(sTV[i])
            yield delay(1)

        raise StopSimulation()
    return instances()

TB=MUX2_1_B_TBV()
TB.convert(hdl="Verilog", initial_values=True)
VerilogTextReader('MUX2_1_B_TBV');
```

```
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
***Verilog modular from MUX2_1_B_TBV.v***
```

```
// File: MUX2_1_B_TBV.v
// Generated by MyHDL 0.10
// Date: Sat Sep 22 02:13:03 2018
```

```
`timescale 1ns/10ps
```

```
module MUX2_1_B_TBV (
```

```
);
```

```
// myHDL -> Verilog testbench for module `MUX2_1_B`
```

```
reg x0 = 0;
```

```

reg x1 = 0;
reg s = 0;
reg y = 0;
wire [17:0] x0TV;
wire [17:0] x1TV;
wire [17:0] sTV;

assign x0TV = 18'd52583;
assign x1TV = 18'd87798;
assign sTV = 18'd16277;

always @(x0, x1, s, y) begin: MUX2_1_B_TBV_PRINT_DATA
    $write("%h", x0);
    $write(" ");
    $write("%h", x1);
    $write(" ");
    $write("%h", s);
    $write(" ");
    $write("%h", y);
    $write("\n");
end

always @(s, x1, x0) begin: MUX2_1_B_TBV_MUX2_1_B0_0_LOGIC
    if (s) begin
        y = x1;
    end
    else begin
        y = x0;
    end
end

initial begin: MUX2_1_B_TBV_STIMULES
    integer i;
    for (i=0; i<18; i=i+1) begin
        x0 <= x0TV[i];
        x1 <= x1TV[i];
        s <= sTV[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
/home/iridium/anaconda3/lib/python3.6/site-packages/myhdl/conversion/_t
oVerilog.py:349: ToVerilogWarning: Signal is not driven: sTV
    category=ToVerilogWarning

```

▼ 9.5 PYNQ-Z1 Deployment

▼ 9.5.1 Board Circuit

See Board Circuit for "2 Channel Input:1 Channel Output multiplexer in Gate Level Logic"

▼ 9.5.2 Board Constraint

uses the same MUX2_1.xdc as "2 Channel Input:1 Channel Output multiplexer in Gate Level Logic"

▼ 9.5.3 Video of Deployment

MUX2_1_B myHDL PYNQ-Z1 ([YouTube \(https://www.youtube.com/watch?v=QrHQfx_Sjnw\)](https://www.youtube.com/watch?v=QrHQfx_Sjnw))

▼ 10 4:1 MUX via Behavioral if-elif-else

▼ 10.1 myHDL Module

```
In [52]: @block
▼ def MUX4_1_B(x0, x1, x2, x3, s0, s1, y):
    """
    4:1 Multiplexer written in if-elif-else Behavioral
    Input:
    x0(bool): input channel 0
    x1(bool): input channel 1
    x2(bool): input channel 2
    x3(bool): input channel 3
    s1(bool): channel selection input bit 1
    s0(bool): channel selection input bit 0
    Output:
    y(bool): output
    """

    @always_comb
    ▼ def logic():
    ▼     if s0==0 and s1==0:
    ▼         y.next=x0
    ▼     elif s0==1 and s1==0:
    ▼         y.next=x1
    ▼     elif s0==0 and s1==1:
    ▼         y.next=x2
    ▼     else:
    ▼         y.next=x3

    return instances()
```


▼ 10.2 myHDL Testing

```

In [53]: ▾ #generate systmatic and random test values
TestLen=5
SystmaticVals=list(itertools.product([0,1], repeat=6))

s0TVs=np.array([i[0] for i in SystmaticVals]).astype(int)
np.random.seed(15)
s0TVs=np.append(s0TVs, np.random.randint(0,2, TestLen)).astype(int)

s1TVs=np.array([i[1] 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)
s1TVs=np.append(s1TVs, np.random.randint(0,2, TestLen)).astype(int)

x0TVs=np.array([i[2] 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(17)
x0TVs=np.append(x0TVs, np.random.randint(0,2, TestLen)).astype(int)

x1TVs=np.array([i[3] 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(18)
x1TVs=np.append(x1TVs, np.random.randint(0,2, TestLen)).astype(int)

x2TVs=np.array([i[4] 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(19)
x2TVs=np.append(x2TVs, np.random.randint(0,2, TestLen)).astype(int)

x3TVs=np.array([i[5] 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(20)
x3TVs=np.append(x3TVs, np.random.randint(0,2, TestLen)).astype(int)

TestLen=len(x0TVs)
SystmaticVals, s0TVs, s1TVs, x3TVs, x2TVs, x1TVs, x0TVs, TestLen

```

```

Out[53]: ([ (0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 1),
(0, 0, 0, 0, 1, 0),
(0, 0, 0, 0, 1, 1),
(0, 0, 0, 1, 0, 0),
(0, 0, 0, 1, 0, 1),
(0, 0, 0, 1, 1, 0),
(0, 0, 0, 1, 1, 1),
(0, 0, 1, 0, 0, 0),
(0, 0, 1, 0, 0, 1),
(0, 0, 1, 0, 1, 0),
(0, 0, 1, 0, 1, 1),

```

```
(0, 0, 1, 1, 0, 0),
(0, 0, 1, 1, 0, 1),
(0, 0, 1, 1, 1, 0),
(0, 0, 1, 1, 1, 1),
(0, 1, 0, 0, 0, 0),
(0, 1, 0, 0, 0, 1),
(0, 1, 0, 0, 1, 0),
(0, 1, 0, 0, 1, 1)
```

In [54]:

```
Peeker.clear()
x0=Signal(bool(0)); Peeker(x0, 'x0')
x1=Signal(bool(0)); Peeker(x1, 'x1')
x2=Signal(bool(0)); Peeker(x2, 'x2')
x3=Signal(bool(0)); Peeker(x3, 'x3')

s0=Signal(bool(0)); Peeker(s0, 's0')
s1=Signal(bool(0)); Peeker(s1, 's1')
y=Signal(bool(0)); Peeker(y, 'y')

DUT=MUX4_1_B(x0, x1, x2, x3, s0, s1, y)

def MUX4_1_B_TB():
    """
    myHDL only testbench for module `MUX4_1_B`
    """

    @instance
    def stimules():
        for i in range(TestLen):
            x0.next=int(x0TVs[i])
            x1.next=int(x1TVs[i])
            x2.next=int(x2TVs[i])
            x3.next=int(x3TVs[i])
            s0.next=int(s0TVs[i])
            s1.next=int(s1TVs[i])

            yield delay(1)

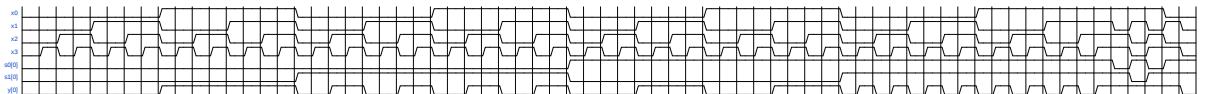
        raise StopSimulation()

    return instances()

sim=Simulation(DUT, MUX4_1_B_TB(), *Peeker.instances()).run()
```

In [55]:

```
Peeker.to_wavedrom()
```



```
In [56]: MUX4_1_BData=Peeker.to_dataframe()
MUX4_1_BData=MUX4_1_BData[['x3', 'x2', 'x1', 'x0', 's1', 's0', 'y']]
MUX4_1_BData
```

Out[56]:

	x3	x2	x1	x0	s1	s0	y
0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0
2	0	1	0	0	0	0	0
3	1	1	0	0	0	0	0
4	0	0	1	0	0	0	0
5	1	0	1	0	0	0	0
6	0	1	1	0	0	0	0
7	1	1	1	0	0	0	0
8	0	0	0	1	0	0	1
9	1	0	0	1	0	0	1
10	0	1	0	1	0	0	1

```
In [57]: Test=MUX4_1_ComboData[['x3', 'x2', 'x1', 'x0', 's1', 's0', 'y']]==MUX4_
Test=Test.all().all()
print(f'Module `MUX4_1_B` works as expected: {Test}')
```

Module `MUX4_1_B` works as expected: True

▼ 10.3 Verilog Conversion

```
In [58]: DUT.convert()  
VerilogTextReader('MUX4_1_B');
```

```
***Verilog modular from MUX4_1_B.v***
```

```
// File: MUX4_1_B.v  
// Generated by MyHDL 0.10  
// Date: Sat Sep 22 02:13:32 2018
```

```
`timescale 1ns/10ps
```

```
module MUX4_1_B (  
    x0,  
    x1,  
    x2,  
    x3,  
    s0,  
    s1,  
    y  
);  
// 4:1 Multiplexer written in if-elif-else Behavioral  
// Input:  
//    x0(bool): input channel 0  
//    x1(bool): input channel 1  
//    x2(bool): input channel 2  
//    x3(bool): input channel 3  
//    s1(bool): channel selection input bit 1  
//    s0(bool): channel selection input bit 0  
// Output:  
//    y(bool): output
```

```
input x0;  
input x1;  
input x2;  
input x3;  
input s0;  
input s1;  
output y;  
reg y;
```

```
always @(x1, x2, s1, s0, x0, x3) begin: MUX4_1_B_LOGIC  
    if (((s0 == 0) && (s1 == 0))) begin  
        y = x0;  
    end  
    else if (((s0 == 1) && (s1 == 0))) begin  
        y = x1;  
    end  
    else if (((s0 == 0) && (s1 == 1))) begin  
        y = x2;  
    end  
    else begin  
        y = x3;  
    end  
end
```

end

endmodule

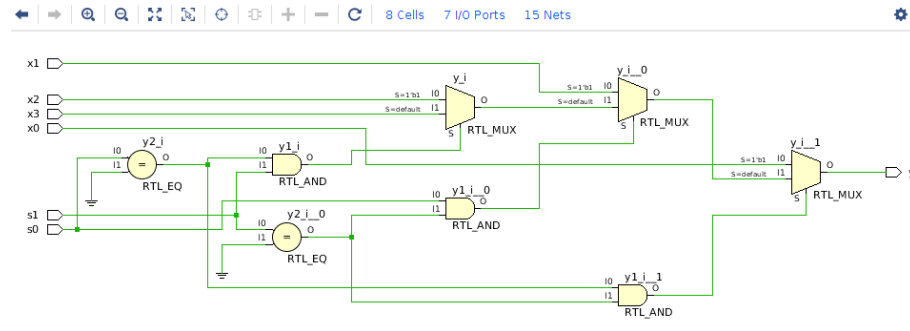


Figure 17: MUX4_1_B RTL schematic; Xilinx Vivado 2017.4

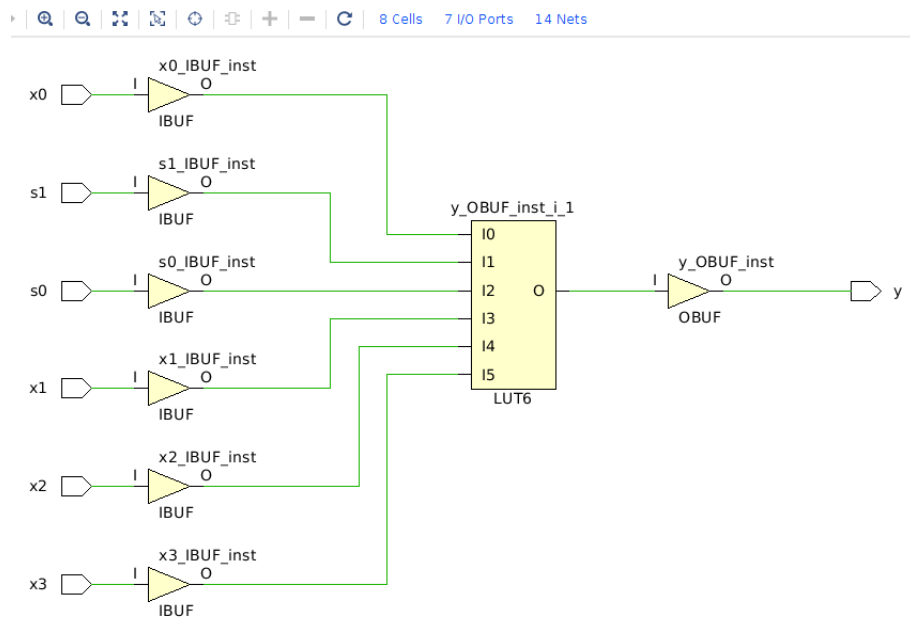


Figure 18: MUX4_1_B Synthesized Schematic; Xilinx Vivado 2017.4

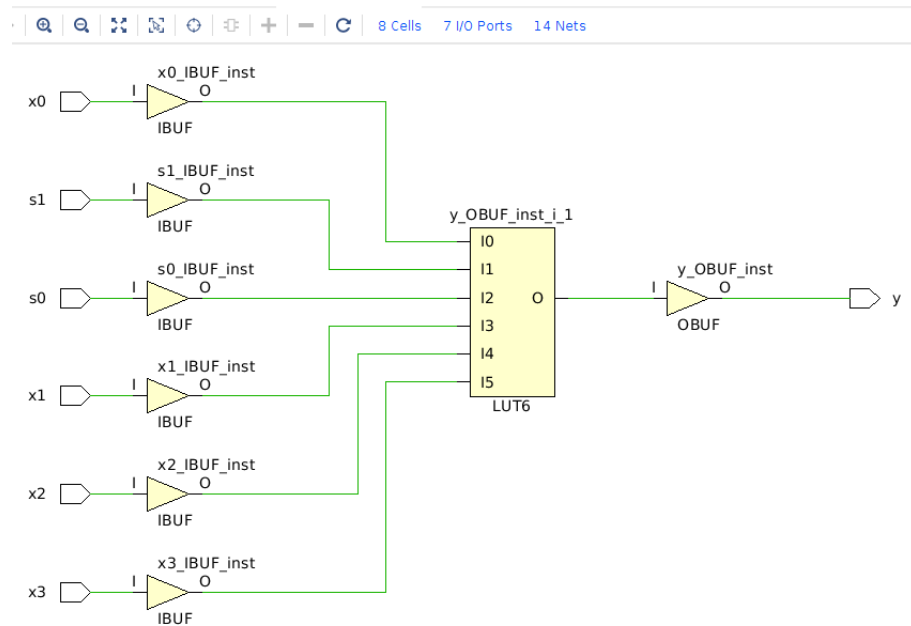


Figure 19: MUX4_1_B Implemented Schematic; Xilinx Vivado 2017.4

10.4 myHDL to Verilog Testbench

```
In [59]: #create BitVectors for MUX4_1_Combo_TBV
x0TVs=intbv(int(''.join(x0TVs.astype(str)), 2))[TestLen:]
x1TVs=intbv(int(''.join(x1TVs.astype(str)), 2))[TestLen:]
x2TVs=intbv(int(''.join(x2TVs.astype(str)), 2))[TestLen:]
x3TVs=intbv(int(''.join(x3TVs.astype(str)), 2))[TestLen:]

s0TVs=intbv(int(''.join(s0TVs.astype(str)), 2))[TestLen:]
s1TVs=intbv(int(''.join(s1TVs.astype(str)), 2))[TestLen:]

x0TVs, bin(x0TVs), x1TVs, bin(x1TVs), x2TVs, bin(x2TVs), x3TVs, bin(x3TVs)
```

```
Out[59]: (intbv(2296870857426870268),
'111111110000000011111111000000001111111100000000111111111100',
intbv(34723282962276803050),
'11110000111100001111000011110000111100001111000011110000111101010',
intbv(118059162071741130356),
'110011001100110011001100110011001100110011001100110011001100110100',
intbv(196765270119568550582),
'1010101010101010101010101010101010101010101010101010101010101011011
0',
intbv(137438953451),
'111111111111111111111111111111111101011',
intbv(9007061817884663),
'11111111111111111000000000000000111111111111111110111')
```

```

In [60]: @block
def MUX4_1_B_TBV():
    """
    myHDL -> Verilog testbench for module `MUX4_1_B`
    """

    x0=Signal(bool(0))
    x1=Signal(bool(0))
    x2=Signal(bool(0))
    x3=Signal(bool(0))
    y=Signal(bool(0))
    s0=Signal(bool(0))
    s1=Signal(bool(0))

    @always_comb
    def print_data():
        print(x0, x1, x2, x3, s0, s1, y)

    #Test Signal Bit Vectors
    x0TV=Signal(x0TVs)
    x1TV=Signal(x1TVs)
    x2TV=Signal(x2TVs)
    x3TV=Signal(x3TVs)
    s0TV=Signal(s0TVs)
    s1TV=Signal(s1TVs)

    DUT=MUX4_1_B(x0, x1, x2, x3, s0, s1, y)

    @instance
    def stimules():
        for i in range(TestLen):
            x0.next=int(x0TV[i])
            x1.next=int(x1TV[i])
            x2.next=int(x2TV[i])
            x3.next=int(x3TV[i])
            s0.next=int(s0TV[i])
            s1.next=int(s1TV[i])
            yield delay(1)

        raise StopSimulation()
    return instances()

TB=MUX4_1_B_TBV()
TB.convert(hdl="Verilog", initial_values=True)
VerilogTextReader('MUX4_1_B_TBV');

```

```

<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>
<class 'myhdl._Signal._Signal'> <class '_ast.Name'>

```



```
// File: MUX4_1_B_TBV.v
// Generated by MyHDL 0.10
// Date: Sat Sep 22 02:13:42 2018
```

```
`timescale 1ns/10ps
```

```
module MUX4_1_B_TBV (
```

```
);
```

```
// MUX4_1_B_TBV is a 4:1 multiplexer in Gate Level Logic
```

▼ 10.5 PYNQ-Z1 Deployment

▼ 10.5.1 Board Circuit

See Board Circuit for "4 Channel Input : 1 Channel Output multiplexer in Gate Level Logic"

▼ 10.5.2 Board Constraint

uses same 'MUX4_1.xdc' as "4 Channel Input : 1 Channel Output multiplexer in Gate Level Logic"

▼ 10.5.3 Video of Deployment

MUX4_1_B myHDL PYNQ-Z1 ([YouTube \(https://www.youtube.com/watch?v=UKXx4PYS1xI\)](https://www.youtube.com/watch?v=UKXx4PYS1xI))

▼ 11 Multiplexer 4:1 Behavioral via Bitvectors

▼ 11.1 myHDL Module

```
In [61]: @block
def MUX4_1_BV(X, S, y):
    """
    4:1 Multiplexer written in behavioral "if-elif-else"(case)
    with BitVector inputs
    Input:
        X(4bitBV):input bit vector; min=0, max=15
        S(2bitBV):selection bit vector; min=0, max=3
    Output:
        y(bool): ouput
    """

    @always_comb
    def logic():
        if S==0:
            y.next=X[0]
        elif S==1:
            y.next=X[1]
        elif S==2:
            y.next=X[2]
        else:
            y.next=X[3]

    return instances()
```

▼ 11.2 myHDL Testing

```
In [62]: XTVs=np.array([1,2,4,8])
XTVs=np.append(XTVs, np.random.choice([1,2,4,8], 6)).astype(int)
TestLen=len(XTVs)

np.random.seed(12)
STVs=np.arange(0,4)
STVs=np.append(STVs, np.random.randint(0,4, 5))
TestLen, XTVs, STVs
```

```
Out[62]: (10, array([1, 2, 4, 8, 4, 2, 1, 8, 4, 8]), array([0, 1, 2, 3, 3, 3, 2,
1, 1]))
```

```
In [63]: Peeker.clear()
X=Signal(intbv(0)[4:]); Peeker(X, 'X')
S=Signal(intbv(0)[2:]); Peeker(S, 'S')
y=Signal(bool(0)); Peeker(y, 'y')

DUT=MUX4_1_BV(X, S, y)

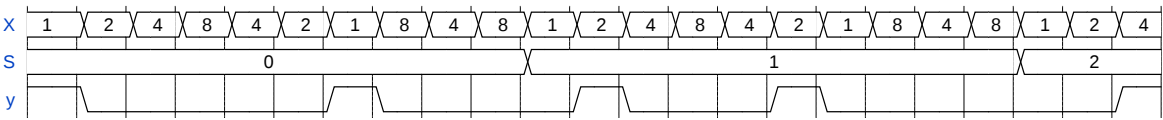
▼ def MUX4_1_BV_TB():
    @instance
    ▼ def stimules():
        ▼ for i in STVs:
            ▼ for j in XTVs:
                S.next=int(i)
                X.next=int(j)
                yield delay(1)

            raise StopSimulation()

    return instances()

sim=Simulation(DUT, MUX4_1_BV_TB(), *Peeker.instances()).run()
```

```
In [64]: Peeker.to_wavedrom('X', 'S', 'y', start_time=0, stop_time=2*TestLen+2)
```



```
In [65]: MUX4_1_BVData=Peeker.to_dataframe()
MUX4_1_BVData=MUX4_1_BVData[['X', 'S', 'y']]
MUX4_1_BVData
```

Out[65]:

	X	S	y
0	1	0	1
1	2	0	0
2	4	0	0
3	8	0	0
4	4	0	0
5	2	0	0
6	1	0	1
7	8	0	0
8	4	0	0
9	8	0	0
10	1	1	0

```
In [66]: MUX4_1_BVData['x0']=None; MUX4_1_BVData['x1']=None; MUX4_1_BVData['x2']
MUX4_1_BVData[['x3', 'x2', 'x1', 'x0']]=MUX4_1_BVData[['X']].apply(lambk

MUX4_1_BVData['s0']=None; MUX4_1_BVData['s1']=None
MUX4_1_BVData[['s1', 's0']]=MUX4_1_BVData[['S']].apply(lambda bv: [int(

MUX4_1_BVData=MUX4_1_BVData[['X', 'x0', 'x1', 'x2', 'x3', 'S', 's0', 's
MUX4_1_BVData
```

Out[66]:

	X	x0	x1	x2	x3	S	s0	s1	y
0	1	1	0	0	0	0	0	0	1
1	2	0	1	0	0	0	0	0	0
2	4	0	0	1	0	0	0	0	0
3	8	0	0	0	1	0	0	0	0
4	4	0	0	1	0	0	0	0	0
5	2	0	1	0	0	0	0	0	0
6	1	1	0	0	0	0	0	0	1
7	8	0	0	0	1	0	0	0	0
8	4	0	0	1	0	0	0	0	0
9	8	0	0	0	1	0	0	0	0
10	1	1	0	0	0	1	1	0	0

```
In [67]: MUX4_1_BVData['yRef']=MUX4_1_BVData.apply(lambda row:y41EqN(row['x0'],
MUX4_1_BVData
```

/home/iridium/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: <http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy> (<http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy>)
 """Entry point for launching an IPython kernel.

```
In [68]: Test=(MUX4_1_BVData['y']==MUX4_1_BVData['yRef']).all()  
print(f'Module `MUX4_1_BVData` works as expected: {Test}')
```

Module `MUX4_1_BVData` works as expected: True

▼ 11.3 Verilog Conversion

```
In [69]: DUT.convert()  
VerilogTextReader('MUX4_1_BV');
```

```
***Verilog modular from MUX4_1_BV.v***
```

```
// File: MUX4_1_BV.v  
// Generated by MyHDL 0.10  
// Date: Sat Sep 22 02:14:13 2018
```

```
`timescale 1ns/10ps
```

```
module MUX4_1_BV (  
    X,  
    S,  
    y  
);  
// 4:1 Multiplexer written in behavioral "if-elif-else"(case)  
// with BitVector inputs  
// Input:  
//     X(4bitBV):input bit vector; min=0, max=15  
//     S(2bitBV):selection bit vector; min=0, max=3  
// Output:  
//     y(bool): output
```

```
input [3:0] X;  
input [1:0] S;  
output y;  
reg y;
```

```
always @(X, S) begin: MUX4_1_BV_LOGIC  
    case (S)  
        'h0: begin  
            y = X[0];  
        end  
        'h1: begin  
            y = X[1];  
        end  
        'h2: begin  
            y = X[2];  
        end  
        default: begin  
            y = X[3];  
        end  
    endcase  
end  
  
endmodule
```

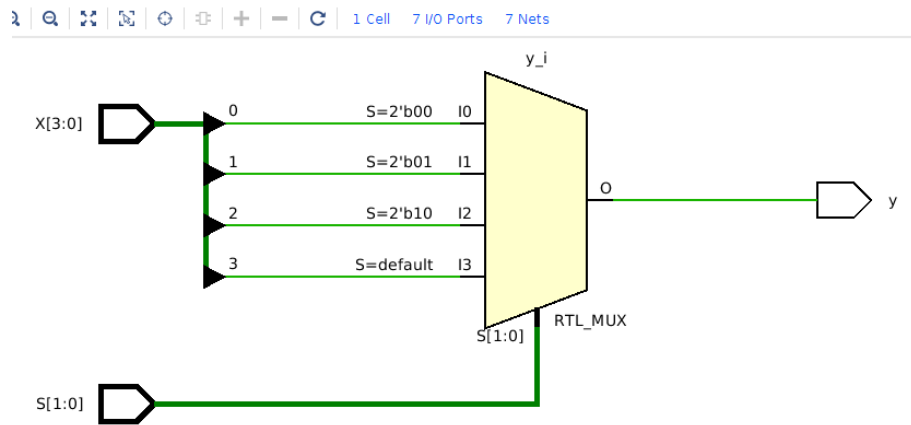


Figure 20: MUX4_1_BV RTL schematic; Xilinx Vivado 2017.4

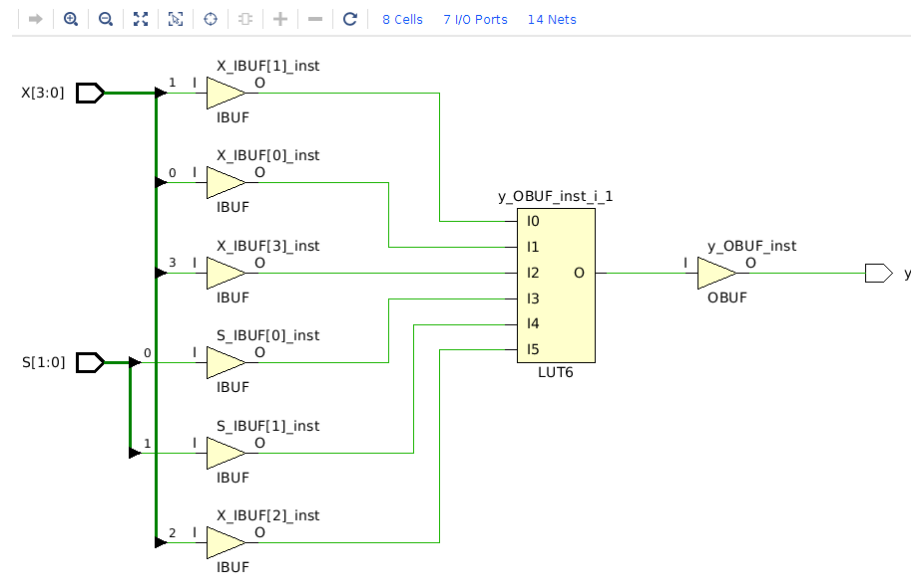


Figure 21: MUX4_1_BV Synthesized Schematic; Xilinx Vivado 2017.4

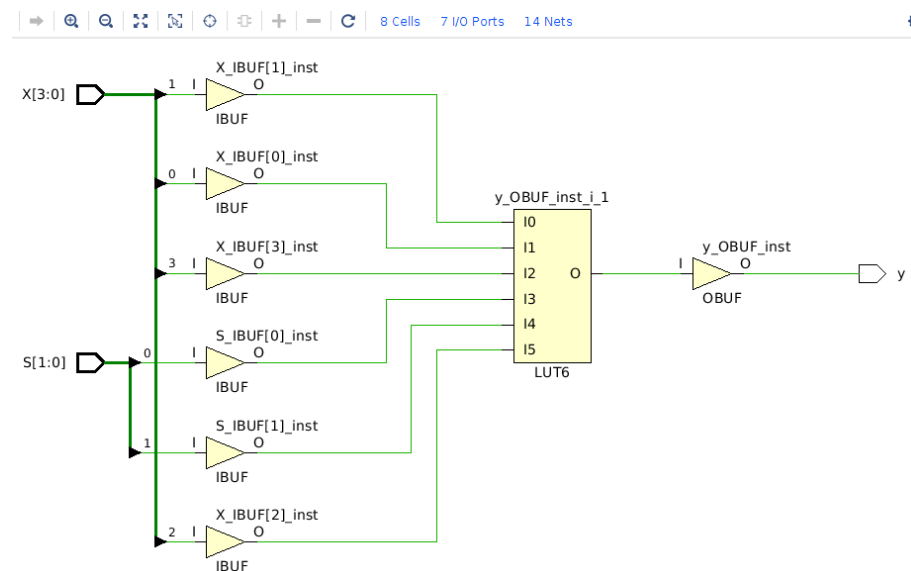


Figure 22: MUX4_1_BV Implemented Schematic; Xilinx Vivado 2017.4

▼ 11.4 myHDL to Verilog Testbench

Will Do later

▼ 11.5 PYNQ-Z1 Deployment

▼ 11.5.1 Board Circuit

See Board Circuit for "4 Channel Input : 1 Channel Output multiplexer in Gate Level Logic"

▼ 11.5.2 Board Constraint

notice that in `get_ports` the pin is set to the a single bit of the bitvector via bitvector indexing

```
In [70]: ConstraintXDCTextReader('MUX4_1_BV');

***Constraint file from MUX4_1_BV.xdc***

## PYNQ-Z1 Constraint File for MUX4_1_BV
## Based on https://github.com/Xilinx/PYNQ/blob/master/boards/Pynq-Z1/base/vivado/constraints/base.xdc (https://github.com/Xilinx/PYNQ/blob/master/boards/Pynq-Z1/base/vivado/constraints/base.xdc)

## Switches
set_property -dict {PACKAGE_PIN M20 IOSTANDARD LVCMOS33} [get_ports {S
[0]}}; ## SW0
set_property -dict {PACKAGE_PIN M19 IOSTANDARD LVCMOS33} [get_ports {S
[1]}}; ## SW1

## Buttons
set_property -dict {PACKAGE_PIN D19 IOSTANDARD LVCMOS33} [get_ports {X
[0]}}; ## BT0
set_property -dict {PACKAGE_PIN D20 IOSTANDARD LVCMOS33} [get_ports {X
[1]}}; ## BT1
set_property -dict {PACKAGE_PIN L20 IOSTANDARD LVCMOS33} [get_ports {X
[2]}}; ## BT2
set_property -dict {PACKAGE_PIN L19 IOSTANDARD LVCMOS33} [get_ports {X
[3]}}; ## BT3

## LEDs
set_property -dict {PACKAGE_PIN R14 IOSTANDARD LVCMOS33} [get_ports
{y}]; ## Led 0
```

▼ 11.5.3 Video of Deployment

MUX4_1_BV myHDL PYNQ-Z1 ([YouTube \(https://www.youtube.com/watch?v=vFG9kgLXJek\)](https://www.youtube.com/watch?v=vFG9kgLXJek))