Shift Registers in myHDL

Steven K Armour

Shift registers are common structures that move around data in primitive memory by rearranging the bit order. While shift registers are built as needed they typically fall into five categories. The four conversion types: PIPO, PISO, SISO, SIPO; and cyclic shift registers such as the ring and Johnson counters. Wich can be used as counters but are in reality shift registers since they reorder the bits in the modules internal memory

Table of Contents

1 Refrances

@misc{myhdl_2017, title={Johnson Counter}, url={http://www.myhdl.org/docs/examples/jc2.html} (http://www.myhdl.org/docs/examples/jc2.html}), journal={Myhdl.org}, author={myHDL}, year= {2017} }

@misc{the shift register, url={https://www.electronics-tutorials.ws/sequential/seq_5.html} (https://www.electronics-tutorials.ws/sequential/seq_5.html}), journal={Electronics Tutorials} },

@misc{petrescu, title={Shift Registers}, url={http://www.csit-sun.pub.ro/courses/Masterat/Xilinx%20Synthesis%20Technology/toolbox.xilinx.com/docsan/xilinx4/ (http://www.csit-sun.pub.ro/courses/Masterat/Xilinx%20Synthesis%20Technology/toolbox.xilinx.com/docsan/xilinx4/ journal={Csit-sun.pub.ro}, author={Petrescu, Adrian} },

@misc{reddy_2014, title={verilog code for ALU,SISO,PIPO,SIPO,PISO}, url= {http://thrinadhreddy.blogspot.com/2014/01/verilog-code-for-alusisopiposipopiso.html} (http://thrinadhreddy.blogspot.com/2014/01/verilog-code-for-alusisopiposipopiso.html}), journal= {Thrinadhreddy.blogspot.com}, author={Reddy, Trinadh}, year={2014} }

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 random

#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
random	The 'random' distribution was not found and is required by the application
	Wed Sep 05 07:50:11 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]:
```
CountVal=17
BitSize=int(np.log2(CountVal))+1; BitSize
```

Out[3]: 5

3 Parallel-In Parallel-Out (PIPO) Shift Register

A PIPO shift Register is one of the most redundant of the four classic shift registers when used as a One Bus In, *One* Bus Out, thus the PIPO here is implemented as a One Bus In, *Two* Bus Out. Further, this opportunity is taken here to talk about HDL algorithms vs HDL Implementation by presenting the same One In, Two Out PIPO but implemented as an asynchronous case and two synchronous cases. While there are obvious differences algorithmically due to the asynchronous vs synchronous. The hardware implementation is even more strikingly different and serves as case in point that HDL programming is neither hardware or software but an intermedte between the worlds. But with grave consequences when translated into hardware that the HDL must keep in perspective when writing Hardware Descriptive Language code.

In [4]:
```
TestData=np.random.randint(0, 2**4, 15); TestData
```

Out[4]:
```
array([ 2,  7,  8, 11,  0, 14,  1,  5, 12,  1, 13,  8,  2, 11,  6])
```

3.1 Asynchronous

In [5]:

```
@block
def PIPO_AS1(DataIn, DataOut1, DataOut2):
    """
    1:2 PIPO shift regestor with no clock (Asynchronous)

    Input:
        DataIn(bitvec)
    Output:
        DataOut1(bitVec): ouput one bitvec len should be same as
            `DataIn`
        DataOut2(bitVec):ouput two bitvec len should be same as
            `DataIn`
    """
    @always_comb
    def logic():
        DataOut1.next=DataIn
        DataOut2.next=DataIn

    return instances()
```

3.1.1 myHDL Testing

In [6]:

```
Peeker.clear()
clk=Signal(bool(0)); Peeker(clk, 'clk')
rst=Signal(bool(0)); Peeker(rst, 'rst')
DataIn=Signal(intbv(0)[4:]); Peeker(DataIn, 'DataIn')
DataOut1=Signal(intbv(0)[4:]); Peeker(DataOut1, 'DataOut1')
DataOut2=Signal(intbv(0)[4:]); Peeker(DataOut2, 'DataOut2')

DUT=PIPO_AS1(DataIn, DataOut1, DataOut2)

def PIPO_TB():
    """
    myHDL only Testbench for `PIPO_*` module
    """
    @always(delay(1))
    def ClkGen():
        clk.next=not clk

    @instance
    def stimules():
        i=0
        while True:

            DataIn.next=int(TestData[i])

            if i==14:
                raise StopSimulation()


            i+=1
            yield clk.posedge

    return instances()

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

In [7]:

```
Peeker.to_wavedrom()
```

In [8]:

```
PIPO_AS1Data=Peeker.to_dataframe();
PIPO_AS1Data=PIPO_AS1Data[PIPO_AS1Data['clk']==1]
PIPO_AS1Data.drop(['clk', 'rst'], axis=1, inplace=True)
PIPO_AS1Data.reset_index(drop=True, inplace=True)
PIPO_AS1Data
```

Out[8]:

	DataIn	DataOut1	DataOut2
0	7	7	7
1	8	8	8
2	11	11	11
3	0	0	0
4	14	14	14
5	1	1	1
6	5	5	5
7	12	12	12
8	1	1	1
9	13	13	13
10	8	8	8
11	2	2	2
12	11	11	11

3.1.2 Verilog Code

In [9]:
```
DUT.convert()
VerilogTextReader('PIPO_AS1');
```

```
***Verilog modual from PIPO_AS1.v***

 // File: PIPO_AS1.v
// Generated by MyHDL 0.10
// Date: Wed Sep  5 07:50:17 2018


`timescale 1ns/10ps

module PIPO_AS1 (
    DataIn,
    DataOut1,
    DataOut2
);
// 1:2 PIPO shift regestor with no clock (Asynchronous)
//
// Input:
//     DataIn(bitvec)
// Output:
//     DataOut1(bitVec): ouput one bitvec len should be same as
//         `DataIn`
//     DataOut2(bitVec):ouput two bitvec len should be same as
//         `DataIn`

input [3:0] DataIn;
output [3:0] DataOut1;
wire [3:0] DataOut1;
output [3:0] DataOut2;
wire [3:0] DataOut2;




assign DataOut1 = DataIn;
assign DataOut2 = DataIn;

endmodule
```

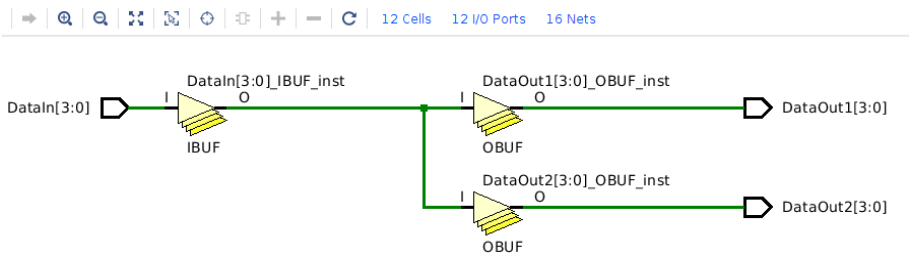

Figure 1: PIPO_AS1 Shift Register RTL schematic; Xilinx Vivado 2017.4

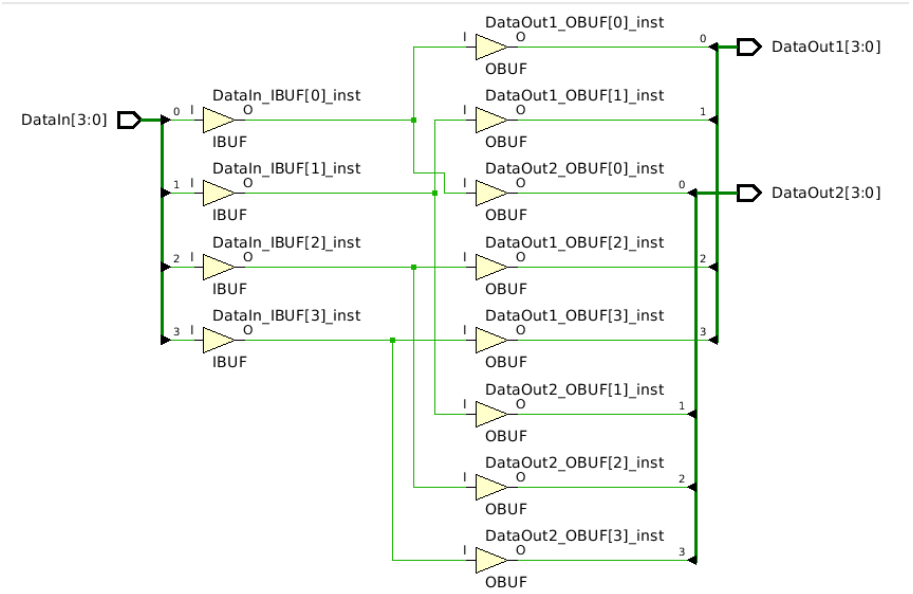

Figure 2: PIPO_AS1 Shift Register Synthesized Schematic; Xilinx Vivado 2017.4

3.1.3 Verilog Testbench (!ToDo)

3.2 Synchronous 1

3.2.1 myHDL Module

In [10]:

```
@block
def PIPO_S1(DataIn, DataOut1, DataOut2, clk, rst):
    """
    one-in two-out PIPO typicaly found in the lititure
    lacking buffering

    Inputs:
        DataIn(bitVec): one-in Parallel data int
        clk(bool): clock
        rst(bool): reset

    Ouputs:
        DataOut1(bitVec): Parallel out 1
        DataOut2(bitVec): Parallel out 1

    """


    @always(clk.posedge, rst.negedge)
    def logic():
        if rst:
            DataOut1.next=0
            DataOut2.next=0
        else:
            DataOut1.next=DataIn
            DataOut2.next=DataIn

    return instances()
```

3.2.2 myHDL Testing

In [11]:

```
Peeker.clear()
clk=Signal(bool(0)); Peeker(clk, 'clk')
rst=Signal(bool(0)); Peeker(rst, 'rst')
DataIn=Signal(intbv(0)[4:]); Peeker(DataIn, 'DataIn')
DataOut1=Signal(intbv(0)[4:]); Peeker(DataOut1, 'DataOut1')
DataOut2=Signal(intbv(0)[4:]); Peeker(DataOut2, 'DataOut2')

DUT=PIPO_S1(DataIn, DataOut1, DataOut2, clk, rst)

def PIPO_TB():
    """
    myHDL only Testbench for `RingCounter` module
    """
    @always(delay(1))
    def ClkGen():
        clk.next=not clk

    @instance
    def stimules():
        i=0
        while True:

            DataIn.next=int(TestData[i])

            if i==14:
                raise StopSimulation()


            i+=1
            yield clk.posedge

    return instances()

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

In [12]:

```
Peeker.to_wavedrom()
```

In [13]:

```
PIPO_S1Data=Peeker.to_dataframe();
PIPO_S1Data=PIPO_S1Data[PIPO_S1Data['clk']==1]
PIPO_S1Data.drop(['clk', 'rst'], axis=1, inplace=True)
PIPO_S1Data.reset_index(drop=True, inplace=True)
PIPO_S1Data
```

Out[13]:

	DataIn	DataOut1	DataOut2
0	7	2	2
1	8	7	7
2	11	8	8
3	0	11	11
4	14	0	0
5	1	14	14
6	5	1	1
7	12	5	5
8	1	12	12
9	13	1	1
10	8	13	13
11	2	8	8
12	11	2	2

3.2.3 Verilog Code

In [14]:
```
DUT.convert()
VerilogTextReader('PIPO_S1');
```

Verilog modual from PIPO_S1.v

```
 // File: PIPO_S1.v
// Generated by MyHDL 0.10
// Date: Wed Sep  5 07:50:34 2018


`timescale 1ns/10ps

module PIPO_S1 (
    DataIn,
    DataOut1,
    DataOut2,
    clk,
    rst
);
// one-in two-out PIPO typicaly found in the lititure
// lacking buffering
//
// Inputs:
//     DataIn(bitVec): one-in Parallel data int
//     clk(bool): clock
//     rst(bool): reset
//
// Ouputs:
//     DataOut1(bitVec): Parallel out 1
//     DataOut2(bitVec): Parallel out 1
//

input [3:0] DataIn;
output [3:0] DataOut1;
reg [3:0] DataOut1;
output [3:0] DataOut2;
reg [3:0] DataOut2;
input clk;
input rst;




always @(posedge clk, negedge rst) begin: PIPO_S1_LOGIC
    if (rst) begin
        DataOut1 <= 0;
        DataOut2 <= 0;
    end
    else begin
        DataOut1 <= DataIn;
        DataOut2 <= DataIn;
    end
end

endmodule
```

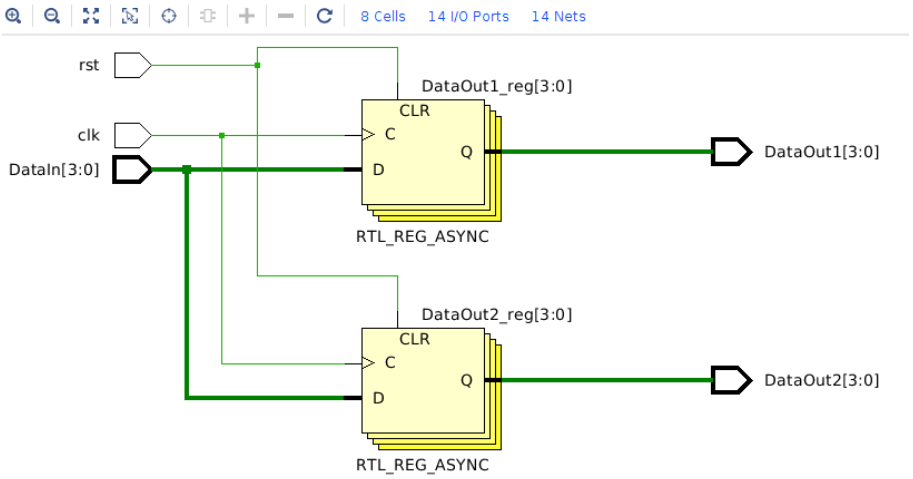

Figure 3: PIPO_S1 Shift Register RTL schematic; Xilinx Vivado 2017.4

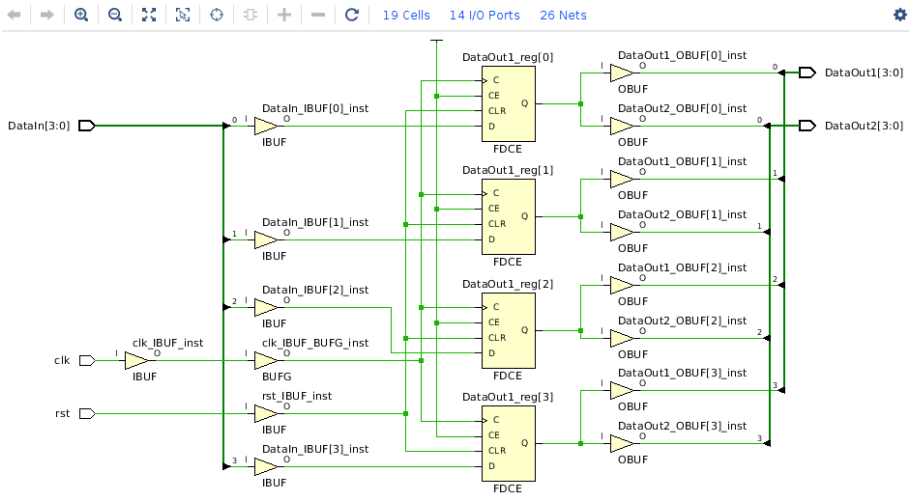

Figure 4: PIPO_S1 Shift Register Synthesized Schematic; Xilinx Vivado 2017.4

3.3 Synchronous 2

3.3.1 myHDL Module

In [15]:

```
@block
def PIPO_S2(DataIn, DataOut1, DataOut2, clk, rst):
    """
    one-in two-out PIPO with buffering

    Inputs:
        DataIn(bitVec): one-in Parallel data int
        clk(bool): clock
        rst(bool): reset

    Ouputs:
        DataOut1(bitVec): Parallel out 1
        DataOut2(bitVec): Parallel out 1

    """

    Buffer=Signal(modbv(0)[len(DataIn):])
    @always(clk.posedge, rst.negedge)
    def logic():
        if rst:
            Buffer.next=0
        else:
            Buffer.next=DataIn


    #not normaly found in PIPO; but is better practice since buffers he
    #with isolation for ASIC desighn
    @always_comb
    def OuputBuffer():
        DataOut1.next=Buffer
        DataOut2.next=Buffer

    return instances()
```

3.3.2 myHDL Testing

In [16]:

```
Peeker.clear()
clk=Signal(bool(0)); Peeker(clk, 'clk')
rst=Signal(bool(0)); Peeker(rst, 'rst')
DataIn=Signal(intbv(0)[4:]); Peeker(DataIn, 'DataIn')
DataOut1=Signal(intbv(0)[4:]); Peeker(DataOut1, 'DataOut1')
DataOut2=Signal(intbv(0)[4:]); Peeker(DataOut2, 'DataOut2')

DUT=PIPO_S2(DataIn, DataOut1, DataOut2, clk, rst)

def PIPO_TB():
    """
    myHDL only Testbench for `PIPO_*` module
    """
    @always(delay(1))
    def ClkGen():
        clk.next=not clk

    @instance
    def stimules():
        i=0
        while True:

            DataIn.next=int(TestData[i])

            if i==14:
                raise StopSimulation()


            i+=1
            yield clk.posedge

    return instances()

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

In [17]:

```
Peeker.to_wavedrom()
```

In [18]:

```
PIPO_S2Data=Peeker.to_dataframe();
PIPO_S2Data=PIPO_S2Data[PIPO_S2Data['clk']==1]
PIPO_S2Data.drop(['clk', 'rst'], axis=1, inplace=True)
PIPO_S2Data.reset_index(drop=True, inplace=True)
PIPO_S2Data
```

Out[18]:

	DataIn	DataOut1	DataOut2
0	7	2	2
1	8	7	7
2	11	8	8
3	0	11	11
4	14	0	0
5	1	14	14
6	5	1	1
7	12	5	5
8	1	12	12
9	13	1	1
10	8	13	13
11	2	8	8
12	11	2	2

3.3.3 Verilog Code

In [19]:
```
DUT.convert()
VerilogTextReader('PIPO_S2');
```

Verilog modual from PIPO_S2.v

```
 // File: PIPO_S2.v
// Generated by MyHDL 0.10
// Date: Wed Sep  5 07:50:41 2018


`timescale 1ns/10ps

module PIPO_S2 (
    DataIn,
    DataOut1,
    DataOut2,
    clk,
    rst
);
// one-in two-out PIPO with buffering
//
// Inputs:
//     DataIn(bitVec): one-in Parallel data int
//     clk(bool): clock
//     rst(bool): reset
//
// Ouputs:
//     DataOut1(bitVec): Parallel out 1
//     DataOut2(bitVec): Parallel out 1
//

input [3:0] DataIn;
output [3:0] DataOut1;
wire [3:0] DataOut1;
output [3:0] DataOut2;
wire [3:0] DataOut2;
input clk;
input rst;

reg [3:0] Buffer;



always @(posedge clk, negedge rst) begin: PIPO_S2_LOGIC
    if (rst) begin
        Buffer <= 0;
    end
    else begin
        Buffer <= DataIn;
    end
end



assign DataOut1 = Buffer;
assign DataOut2 = Buffer;
```

```
endmodule
```

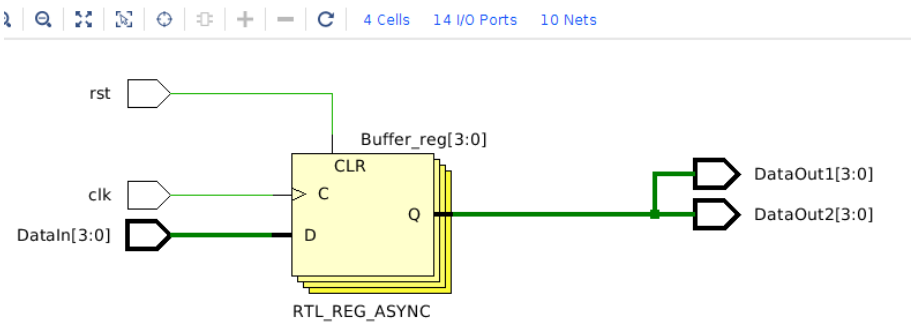

Figure 5: PIPO_S2 Shift Register RTL schematic; Xilinx Vivado 2017.4

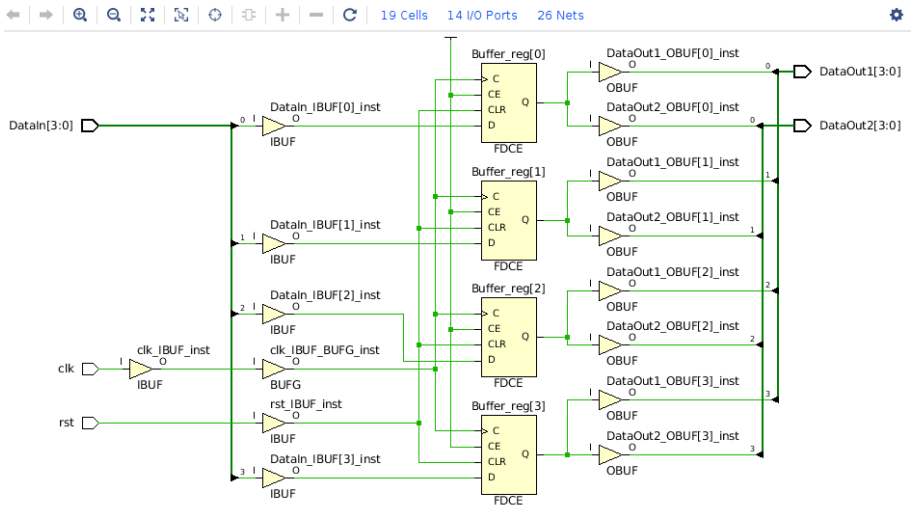

Figure 6: PIPO_S2 Shift Register Synthesized Schematic; Xilinx Vivado 2017.4

3.4 Comparison of the three designs

In the asynchronous case, it can be seen in the RTL that incoming Bus is passed through a buffer and then the Bus is then junctioned to two outputs. This , therefore, does not have any synchronicity to a clock and is the HDL equivalent of taping the wires of the incoming bus to create a copy of the signal. What this design lacks in clock support it gains in resource saving and instantaneous interchange of the input signal to output signals

For the two synchronous cases the incoming signal is buffered by a register set, therefore any signal on the bus will not be present on the output buses for at least one clock cycle. Wich for clocked designs is a good thing but for signals that needs an instantaneous transmission, this is a failing in tradeoff. And while Vivado (and most other FPGA synthesis tools) recognized that the two designs are the same we should not take this for granted. Since other tools may not. Since at the RTL level the implementations are clearly different.

In the first case, the incoming signals is recorded by two registers that then feeds each of the respective output busses. While this means that we gain redundancy in parallel registers. The parallel memories could also become out of sync for any number of reasons. Further, this is a huge amount of resource if it were to be synthesized this way. In comparison, the second design

uses a single register that then feeds each of the outputs where this design has better resource allocation and would not suffer from any asynchronicity between parallel registers. It now suffers in lacking redundancy since any issues in the one register effect both outputs.

The lesson here is that HDL should never be thought of as of programming. Instead, it is a *sophisticated abstraction description* of **Hardware!** And therefore HDL should always be written to satisfy the hardware constraints.

4 Parallel-In Serial-Out (PISO)

Used to translate bus(parallel) data to a single output wire (serial) data a common example is the transmit line of a UART

4.1 myHDL Module

In [20]:
```
@block
def PISO(ReadBus, BusIn, SerialOut, clk, rst):
    """
    Parallel In Serial Out right shift regestor

    Input:
        ReadBus(bool): read bus flag
        BusIn(bool): Serial wire input
        clk(bool): clock
        rst(bool): reset

    Output:
        SerialOut(bool): Serial(wire) output data from `BusIn`
    Note:
        Does not have a finsh serial write indicator
    """

    Buffer=Signal(intbv(0)[len(BusIn):])
    @always(clk.posedge, rst.negedge)
    def logic():
        if rst:
            Buffer.next=0
        elif ReadBus:
            Buffer.next=BusIn
        else:
            Buffer.next=Buffer>>1

    #A more robust PISO would have a counter to trigger
    #a finish serial write flag here

    @always_comb
    def SerialWriteOut():
        SerialOut.next=Buffer[0]

    return instances()
```

4.2 myHDL Testing

In [21]:
```
TestData=np.random.randint(0, 2**4, 3)
print(TestData)
#reverse bit order since right shift
TestDataBin="".join([bin(i, 4)[::-1] for i in TestData])
TestDataBin=[int(i) for i in TestDataBin]
TestDataBin
```

```
[11 15 14]
```

Out[21]: [1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1]

In [22]:

```
Peeker.clear()
ReadBus=Signal(bool(0)); Peeker(ReadBus, 'ReadBus')
BusIn=Signal(intbv(0)[4:]); Peeker(BusIn, 'BusIn')
SerialOut=Signal(bool(0)); Peeker(SerialOut, 'SerialOut')
clk=Signal(bool(0)); Peeker(clk, 'clk')
rst=Signal(bool(0)); Peeker(rst, 'rst')

DUT=PISO(ReadBus, BusIn, SerialOut, clk, rst)

def PISO_TB():
    """
    myHDL only Testbench for `SIPO` module
    """
    @always(delay(1))
    def ClkGen():
        clk.next=not clk

    @instance
    def stimules():
        yield clk.posedge

        for i in TestData:
            BusIn.next=int(i)
            ReadBus.next=True
            yield clk.posedge

            ReadBus.next=False
            for j in range(len(bin(i, 4))):
                yield clk.posedge




        raise StopSimulation()


    return instances()

sim=Simulation(DUT, PISO_TB(), *Peeker.instances()).run()
print(TestData)
print(TestDataBin)
```

```
[11 15 14]
[1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1]
```

In [23]:

```
Peeker.to_wavedrom()
```

In [24]:

```
PISOData=Peeker.to_dataframe();
PISOData=PISOData[PISOData['clk']==1]
PISOData.drop(['clk', 'rst'], axis=1, inplace=True)
PISOData['BusBits']=PISOData['BusIn'].apply(lambda x:bin(x, 4))
PISOData.reset_index(drop=True, inplace=True)
PISOData
```

Out[24]:

	BusIn	ReadBus	SerialOut	BusBits
0	11	1	0	1011
1	11	0	1	1011
2	11	0	1	1011
3	11	0	0	1011
4	11	0	1	1011
5	15	1	0	1111
6	15	0	1	1111
7	15	0	1	1111
8	15	0	1	1111
9	15	0	1	1111
10	14	1	0	1110
11	14	0	0	1110
12	14	0	1	1110
13	14	0	1	1110
14	14	0	1	1110

4.3 Verilog Code

In [25]:
```
DUT.convert()
VerilogTextReader('PISO');
```

Verilog modual from PISO.v

```
 // File: PISO.v
// Generated by MyHDL 0.10
// Date: Wed Sep  5 07:50:56 2018


`timescale 1ns/10ps

module PISO (
    ReadBus,
    BusIn,
    SerialOut,
    clk,
    rst
);
// Parallel In Serial Out right shift regestor
//
// Input:
//     ReadBus(bool): read bus flag
//     BusIn(bool): Serial wire input
//     clk(bool): clock
//     rst(bool): reset
//
// Output:
//     SerialOut(bool): Serial(wire) output data from `BusIn`
// Note:
//     Does not have a finsh serial write indicator

input ReadBus;
input [3:0] BusIn;
output SerialOut;
wire SerialOut;
input clk;
input rst;

reg [3:0] Buffer;



always @(posedge clk, negedge rst) begin: PISO_LOGIC
    if (rst) begin
        Buffer <= 0;
    end
    else if (ReadBus) begin
        Buffer <= BusIn;
    end
    else begin
        Buffer <= (Buffer >>> 1);
    end
end
```

```
assign SerialOut = Buffer[0];

endmodule
```

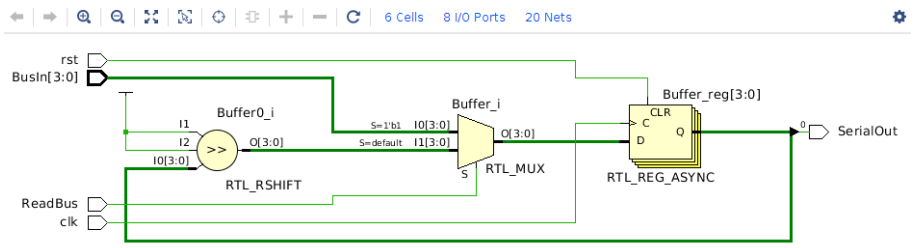

Figure 7: PISO Shift Register RTL schematic; Xilinx Vivado 2017.4

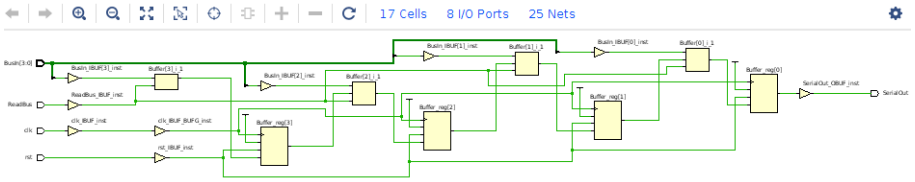

Figure 8: PISO Shift Register Synthesized Schematic; Xilinx Vivado 2017.4

4.4 Verilog Testbench (!ToDo)

5 Serial-In Serial-Out (SISO)

SISO are used to buffer input data like a primitive serial version of the First In First Out (FIFO) Memory. Or can be used for sampling of a wire input that not clock synced to an output that is clock sync

5.1 myHDL Module

In [26]:
```
@block
def SISO(SerialIn, SerialOut, clk, rst, BufferSize):
    """
    SISO Left Shift registor

    Input:
        SerialIn(bool): serial input feed
        clk(bool): clock signal
        rst(bool):reset signal

    Output:
        SerialOut(bool): serial out delayed by BufferSize

    Paramter:
        BufferSize(int): size of SISO buffer, aka delay amount
    """
    Buffer=Signal(modbv(0)[BufferSize:])
    @always(clk.posedge, rst.negedge)
    def logic():
        if rst:
            Buffer.next=0
        else:
            Buffer.next=concat(Buffer[BufferSize-1:0], SerialIn)

    @always_comb
    def ReadLeftMostToSer():
        SerialOut.next=Buffer[BufferSize-1]

    return instances()
```

5.2 myHDL Testing

In [27]:
```
SerialInTVLen=20
np.random.seed(71)
SerialInTV=np.random.randint(0,2,20).astype(int)
SerialInTV
```

Out[27]:
```
array([1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0])
```

In [28]:

```
Peeker.clear()
SerialIn=Signal(bool(0)); Peeker(SerialIn, 'SerialIn')
SerialOut=Signal(bool(0)); Peeker(SerialOut, 'SerialOut')
clk=Signal(bool(0)); Peeker(clk, 'clk')
rst=Signal(bool(0)); Peeker(rst, 'rst')
BufferSize=4

DUT=SISO(SerialIn, SerialOut, clk, rst, BufferSize)

def SISO_TB():
    """
    myHDL only Testbench for `SISO` module
    """
    @always(delay(1))
    def ClkGen():
        clk.next=not clk

    @instance
    def stimules():
        for i in range(SerialInTVLen):
            SerialIn.next=int(SerialInTV[i])
            yield clk.posedge

        for i in range(2):
            if i==0:
                SerialIn.next=0
                rst.next=1
            else:
                rst.next=0
            yield clk.posedge

        for i in range(BufferSize+1):
            SerialIn.next=1
            yield clk.posedge

        raise StopSimulation()

    return instances()

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

In [29]:

```
Peeker.to_wavedrom()
```

In [30]:
```
SISOData=Peeker.to_dataframe()
SISOData
```

Out[30]:

	SerialIn	SerialOut	clk	rst
0	1	0	0	0
1	1	0	1	0
2	1	0	0	0
3	1	0	1	0
4	1	0	0	0
5	1	0	1	0
6	1	0	0	0
7	0	1	1	0
8	0	1	0	0
9	0	1	1	0
10	0	1	0	0
11	0	1	1	0
12	0	1	0	0
13	1	1	1	0
14	1	1	0	0
15	1	0	1	0
16	1	0	0	0
17	0	0	1	0
18	0	0	0	0
19	0	0	1	0
20	0	0	0	0
21	0	1	1	0
22	0	1	0	0
23	1	1	1	0
24	1	1	0	0
25	0	0	1	0
26	0	0	0	0
27	0	0	1	0
28	0	0	0	0
29	1	0	1	0
30	1	0	0	0
31	1	1	1	0
32	1	1	0	0
33	0	0	1	0

	SerialIn	SerialOut	clk	rst
34	0	0	0	0
35	1	0	1	0
36	1	0	0	0
37	0	1	1	0
38	0	1	0	0
39	0	1	1	1
40	0	1	0	1
41	0	0	1	0
42	0	0	0	0
43	1	0	1	0
44	1	0	0	0
45	1	0	1	0
46	1	0	0	0
47	1	0	1	0
48	1	0	0	0
49	1	0	1	0
50	1	0	0	0
51	1	1	1	0
52	1	1	0	0

In [31]:

```
SISOData=SISOData[SISOData['clk']==1]
SISOData.drop('clk', inplace=True, axis=1)
SISOData.reset_index(drop=True, inplace=True)
SISOData
```

```
/home/iridium/anaconda3/lib/python3.6/site-packages/pandas/core/frame.p
y:3697: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-d
ocs/stable/indexing.html#indexing-view-versus-copy (http://pandas.pydat
a.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
  errors=errors)
```

Out[31]:

	SerialIn	SerialOut	rst
0	1	0	0
1	1	0	0
2	1	0	0
3	0	1	0
4	0	1	0
5	0	1	0
6	1	1	0
7	1	0	0
8	0	0	0
9	0	0	0
10	0	1	0
11	1	1	0
12	0	0	0
13	0	0	0
14	1	0	0
15	1	1	0
16	0	0	0
17	1	0	0
18	0	1	0
19	0	1	1
20	0	0	0
21	1	0	0
22	1	0	0
23	1	0	0
24	1	0	0
25	1	1	0

In [32]:
```
SISOData['SerialOutS4']=SISOData['SerialOut'].shift(-4)
SISOData
```

```
/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-d
ocs/stable/indexing.html#indexing-view-versus-copy (http://pandas.pydat
a.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
  """Entry point for launching an IPython kernel.
```

Out[32]:

	SerialIn	SerialOut	rst	SerialOutS4
0	1	0	0	1.0
1	1	0	0	1.0
2	1	0	0	1.0
3	0	1	0	0.0
4	0	1	0	0.0
5	0	1	0	0.0
6	1	1	0	1.0
7	1	0	0	1.0
8	0	0	0	0.0
9	0	0	0	0.0
10	0	1	0	0.0
11	1	1	0	1.0
12	0	0	0	0.0
13	0	0	0	0.0
14	1	0	0	1.0
15	1	1	0	1.0
16	0	0	0	0.0
17	1	0	0	0.0
18	0	1	0	0.0
19	0	1	1	0.0
20	0	0	0	0.0
21	1	0	0	1.0
22	1	0	0	NaN
23	1	0	0	NaN
24	1	0	0	NaN
25	1	1	0	NaN

In [33]:
```
SISOCheck=(SISOData[:16]['SerialIn'] == SISOData[:16]['SerialOutS4'].as
print(f'SISO Check:{SISOCheck}')
```

```
SISO Check:True
```

5.3 Verilog Code

In [34]:
```
DUT.convert()
VerilogTextReader('SISO');
```

Verilog modual from SISO.v

```
 // File: SISO.v
// Generated by MyHDL 0.10
// Date: Wed Sep  5 07:51:08 2018


`timescale 1ns/10ps

module SISO (
    SerialIn,
    SerialOut,
    clk,
    rst
);
// SISO Left Shift registor
//
// Input:
//     SerialIn(bool): serial input feed
//     clk(bool): clock signal
//     rst(bool):reset signal
//
// Output:
//     SerialOut(bool): serial out delayed by BufferSize
//
// Paramter:
//     BufferSize(int): size of SISO buffer, aka delay amount

input SerialIn;
output SerialOut;
wire SerialOut;
input clk;
input rst;

reg [3:0] Buffer;



always @(posedge clk, negedge rst) begin: SISO_LOGIC
    if (rst) begin
        Buffer <= 0;
    end
    else begin
        Buffer <= {Buffer[(4 - 1)-1:0], SerialIn};
    end
end



assign SerialOut = Buffer[(4 - 1)];

endmodule
```

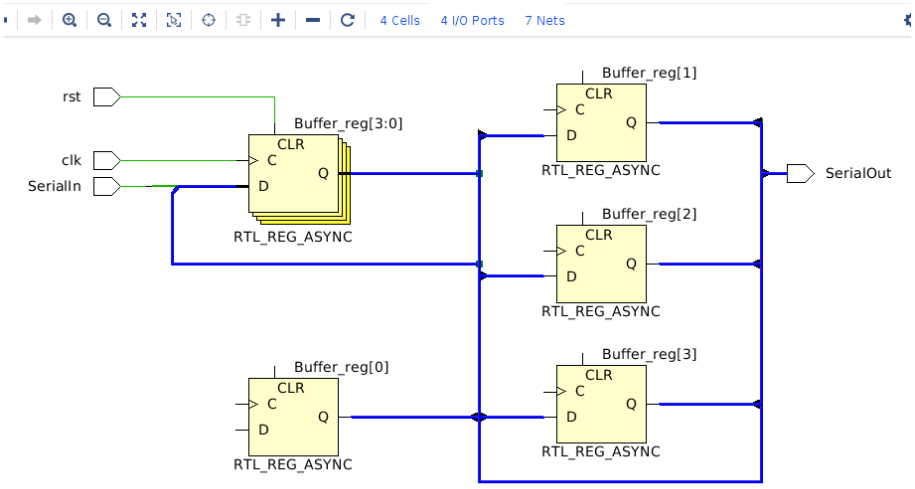

Figure 9: SISO Shift Register RTL schematic; Xilinx Vivado 2017.4

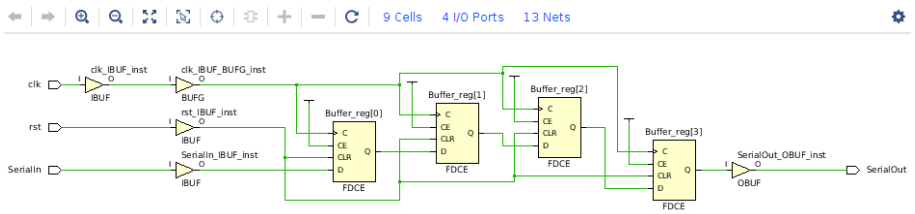

Figure 10: SISO Shift Register Synthesized Schematic; Xilinx Vivado 2017.4

5.4 Verilog Testbench

In [35]:
```
#create BitVector for BufferGate_TBV
SerialInTVs=intbv(int(''.join(SerialInTV.astype(str)), 2))[SerialInTVLe
SerialInTVs, bin(SerialInTVs)
```

Out[35]:
```
(intbv(989338), '11110001100010011010')
```

In [36]:

```
BufferSize=4

@block
def SISO_TBV():
    """
    myHDL -> Verilog Testbench for `SISO` module
    """

    SerialIn=Signal(bool(0))
    SerialOut=Signal(bool(0))
    clk=Signal(bool(0))
    rst=Signal(bool(0))

    @always_comb
    def print_data():
        print(SerialIn, SerialOut, clk, rst)

    SerialInTV=Signal(SerialInTVs)

    DUT=SISO(SerialIn, SerialOut, clk, rst, BufferSize)

    @instance
    def clk_signal():
        while True:
            clk.next = not clk
            yield delay(1)

    @instance
    def stimules():
        for i in range(SerialInTVLen):
            SerialIn.next=int(SerialInTV[i])
            yield clk.posedge

        for i in range(2):
            if i==0:
                SerialIn.next=0
                rst.next=1
            else:
                rst.next=0
            yield clk.posedge

        for i in range(BufferSize+1):
            SerialIn.next=1
            yield clk.posedge

        raise StopSimulation()

    return instances()

TB=SISO_TBV()
TB.convert(hdl="Verilog", initial_values=True)
VerilogTextReader('SISO_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 modual from SISO_TBV.v

```
 // File: SISO_TBV.v
// Generated by MyHDL 0.10
// Date: Wed Sep  5 07:52:33 2018


`timescale 1ns/10ps

module SISO_TBV (

);
// myHDL -> Verilog Testbench for `SISO` module


reg clk = 0;
reg rst = 0;
wire SerialOut;
wire [19:0] SerialInTV;
reg SerialIn = 0;
reg [3:0] SISO0_0_Buffer = 0;

assign SerialInTV = 20'd989338;


always @(rst, SerialOut, SerialIn, clk) begin: SISO_TBV_PRINT_DATA
    $write("%h", SerialIn);
    $write(" ");
    $write("%h", SerialOut);
    $write(" ");
    $write("%h", clk);
    $write(" ");
    $write("%h", rst);
    $write("\n");
end


always @(posedge clk, negedge rst) begin: SISO_TBV_SISO0_0_LOGIC
    if (rst) begin
        SISO0_0_Buffer <= 0;
    end
    else begin
        SISO0_0_Buffer <= {SISO0_0_Buffer[(4 - 1)-1:0], SerialIn};
    end
end



assign SerialOut = SISO0_0_Buffer[(4 - 1)];


initial begin: SISO_TBV_CLK_SIGNAL
    while (1'b1) begin
        clk <= (!clk);
        # 1;
    end
end
```

```
initial begin: SISO_TBV_STIMULES
    integer i;
    for (i=0; i<20; i=i+1) begin
        SerialIn <= SerialInTV[i];
        @(posedge clk);
    end
    for (i=0; i<2; i=i+1) begin
        if ((i == 0)) begin
            SerialIn <= 0;
            rst <= 1;
        end
        else begin
            rst <= 0;
        end
        @(posedge clk);
    end
    for (i=0; i<(4 + 1); i=i+1) begin
        SerialIn <= 1;
        @(posedge clk);
    end
    $finish;
end

endmodule


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

6 Serial-In Parallel-Out (SIPO)

used to translate single wire (serial) data to but (parallel) data, a common example is the read line of a UART

6.1 myHDL Module

In [37]:
```
@block
def SIPO(SerialIn, BusOut, clk, rst):
    """
    Serial In Parallel Out right shift regestor

    Input:
        SerialIn(bool): Serial wire input
        clk(bool): clock
        rst(bool): reset

    Output:
        BusOut(bitVec): Parallel(Bus) output data from `SerialWire`
    """

    Buffer=Signal(modbv(0)[len(BusOut):])
    @always(clk.posedge, rst.negedge)
    def logic():
        if rst:
            Buffer.next=0
        else:
            Buffer.next=concat(Buffer[len(Buffer):0],SerialIn)

    @always_comb
    def OuputBuffer():
        BusOut.next=Buffer

    return instances()
```

6.2 myHDL Testing

In [38]:
```
TestData=np.random.randint(0, 2**4, 3)
print(TestData)
TestDataBin="".join([bin(i, 4) for i in TestData])
TestDataBin=[int(i) for i in TestDataBin]
TestDataBin
```

```
[11 13  0]
```

Out[38]: [1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0]

In [39]:

```
Peeker.clear()
SerialIn=Signal(bool(0)); Peeker(SerialIn, 'SerialIn')
BusOut=Signal(intbv(0)[4:]); Peeker(BusOut, 'BusOut')
clk=Signal(bool(0)); Peeker(clk, 'clk')
rst=Signal(bool(0)); Peeker(rst, 'rst')

DUT=SIPO(SerialIn, BusOut, clk, rst)

def SIPO_TB():
    """
    myHDL only Testbench for `SIPO` module
    """
    @always(delay(1))
    def ClkGen():
        clk.next=not clk

    @instance
    def stimules():
        i=0
        while True:
            if i<len(TestDataBin):
                SerialIn.next=TestDataBin[i]
            elif i==len(TestDataBin):
                pass
            elif i>len(TestDataBin):
                raise StopSimulation()
            i+=1
            yield clk.posedge



        raise StopSimulation()


    return instances()

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

In [40]:

```
Peeker.to_wavedrom()
```

In [41]:
```
SIPOData=Peeker.to_dataframe();
SIPOData=SIPOData[SIPOData['clk']==1]
SIPOData.drop(['clk', 'rst'], axis=1, inplace=True)
SIPOData['BusBits']=SIPOData['BusOut'].apply(lambda x:bin(x, 4))
SIPOData.reset_index(drop=True, inplace=True)
SIPOData
```

Out[41]:

	BusOut	SerialIn	BusBits
0	1	0	0001
1	2	1	0010
2	5	1	0101
3	11	1	1011
4	7	1	0111
5	15	0	1111
6	14	1	1110
7	13	0	1101
8	10	0	1010
9	4	0	0100
10	8	0	1000
11	0	0	0000

In [42]:
```
TestData, TestDataBin
```

Out[42]:
```
(array([11, 13,  0]), [1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0])
```

6.3 Verilog Code

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

```
***Verilog modual from SIPO.v***

 // File: SIPO.v
// Generated by MyHDL 0.10
// Date: Wed Sep  5 07:52:43 2018


`timescale 1ns/10ps

module SIPO (
    SerialIn,
    BusOut,
    clk,
    rst
);
// Serial In Parallel Out right shift regestor
//
// Input:
//     SerialIn(bool): Serial wire input
//     clk(bool): clock
//     rst(bool): reset
//
// Output:
//     BusOut(bitVec): Parallel(Bus) output data from `SerialWire`

input SerialIn;
output [3:0] BusOut;
wire [3:0] BusOut;
input clk;
input rst;

reg [3:0] Buffer = 0;



always @(posedge clk, negedge rst) begin: SIPO_LOGIC
    if (rst) begin
        Buffer <= 0;
    end
    else begin
        Buffer <= {Buffer[4-1:0], SerialIn};
    end
end



assign BusOut = Buffer;

endmodule


/home/iridium/anaconda3/lib/python3.6/site-packages/myhdl/conversion/_t
oVerilog.py:309: ToVerilogWarning: Port is not used: SerialIn
  category=ToVerilogWarning
```

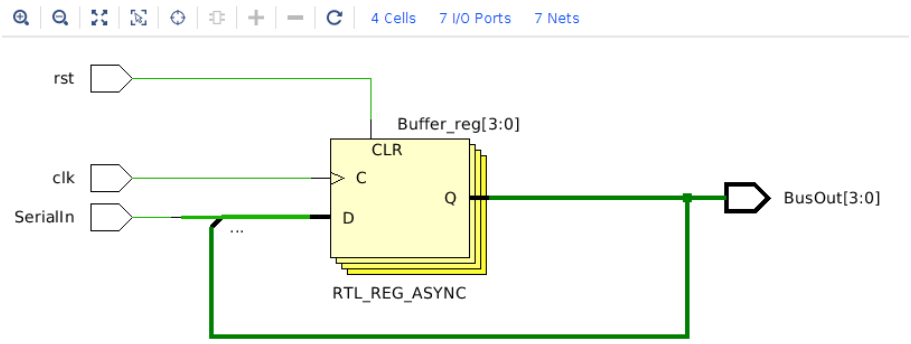

Figure 11: SIPO Shift Register RTL schematic; Xilinx Vivado 2017.4

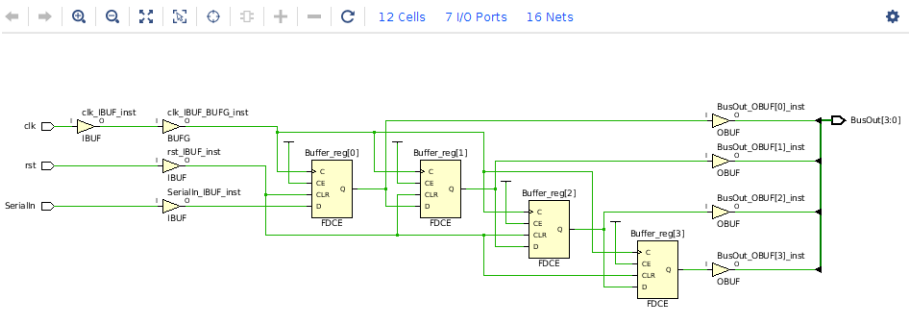

Figure 12: SIPO Shift Register Synthesized Schematic; Xilinx Vivado 2017.4

6.4 Verilog Testbench

In [44]:
```
V=int(''.join([str(i) for i in TestDataBin]), 2)
SerialVal=intbv(V)[len(TestDataBin):]
SerialVal
```

Out[44]:
```
intbv(3024)
```

In [45]:

```
@block
def SIPO_TBV():
    """
    myHDL -> Verilog Testbench for `SIPO` module
    """
    SerialVals=Signal(SerialVal)
    SerialIn=Signal(bool(0))
    BusOut=Signal(intbv(0)[4:])
    clk=Signal(bool(0))
    rst=Signal(bool(0))

    @always_comb
    def print_data():
        print(SerialIn, BusOut, clk, rst)



    DUT=SIPO(SerialIn, BusOut, clk, rst)

    @instance
    def clk_signal():
        while True:
            clk.next = not clk
            yield delay(1)

    @instance
    def stimules():
        i=0
        while True:
            if i<len(TestDataBin):
                SerialIn.next=SerialVals[i]
            elif i==len(TestDataBin):
                pass
            elif i>len(TestDataBin):
                raise StopSimulation()
            i+=1
            yield clk.posedge



        raise StopSimulation()


    return instances()

TB=SIPO_TBV()
TB.convert(hdl="Verilog", initial_values=True)
VerilogTextReader('SIPO_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 modual from SIPO_TBV.v***

 // File: SIPO_TBV.v
// Generated by MyHDL 0.10
```

```
// Date: Wed Sep  5 07:52:49 2018


`timescale 1ns/10ps

module SIPO_TBV (

);
// myHDL -> Verilog Testbench for `SIPO` module


reg clk = 0;
wire rst;
reg SerialIn = 0;
wire [3:0] BusOut;
wire [11:0] SerialVals;
reg [3:0] SIPO0_0_Buffer = 0;

assign rst = 1'd0;
assign SerialVals = 12'd3024;


always @(rst, BusOut, SerialIn, clk) begin: SIPO_TBV_PRINT_DATA
    $write("%h", SerialIn);
    $write(" ");
    $write("%h", BusOut);
    $write(" ");
    $write("%h", clk);
    $write(" ");
    $write("%h", rst);
    $write("\n");
end


always @(posedge clk, negedge rst) begin: SIPO_TBV_SIPO0_0_LOGIC
    if (rst) begin
        SIPO0_0_Buffer <= 0;
    end
    else begin
        SIPO0_0_Buffer <= {SIPO0_0_Buffer[4-1:0], SerialIn};
    end
end



assign BusOut = SIPO0_0_Buffer;


initial begin: SIPO_TBV_CLK_SIGNAL
    while (1'b1) begin
        clk <= (!clk);
        # 1;
    end
end


initial begin: SIPO_TBV_STIMULES
```

```
    integer i;
    i = 0;
    while (1'b1) begin
        if ((i < 12)) begin
            SerialIn <= SerialVals[i];
        end
        else if ((i == 12)) begin
            // pass
        end
        else if ((i > 12)) begin
            $finish;
        end
        i = i + 1;
        @(posedge clk);
    end
    $finish;
end

endmodule
```

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

7 Cyclic Shift Register Johnson Counter

The Johnson Counter is implemented in this notebook about shift registers since in reality, a Johnson Counter is a cyclic shift register with single bit inversion. Hence the Johnson Counter, which goes by the other name of a Mobius Ring Counter. The following are aspects of Johnson Counter by Sougata Bhattacharjee [https://www.quora.com/What-is-the-difference-between-a-Johnson-counter-and-a-ring-counter (https://www.quora.com/What-is-the-difference-between-a-Johnson-counter-and-a-ring-counter)]

- In a Johnson Counter the output bar or Q(bar) of the last flip-flop is connected to the input of the first flip-flop
- If n is the number of the flip-flop used, then the total number of states used is $2n$.
- The Johnson Counter is also known as walking counter, switching tail counter and is mostly used in phase shift or function generator.
- The decoding circuit is complex as compared to a ring counter.
- If input frequency is f in Johnson Counter,then the output is $\frac{f}{2n}$.
- The total number of unused states in Johnson Counter is $(2^n - 2n)$
- The main problem with Johnson counter is that once it enters into an unused state it is in a lockout state

For a four-bit Johnson Counter, the next state diagram is given by the following table from wikipedga (https://en.wikipedia.org/wiki/Ring_counter)

Johnson counter				
State	Q0	Q1	Q2	Q3
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1
0	0	0	0	0

Figure 13: 4-bit Ring Counter next state table from wikipedia (https://en.wikipedia.org/wiki/Ring_counter#Four-bit_ring-counter_sequences)

whereupon examining the next state table for the Johnson counter we can see why a Johnson counter is also called a Mobius counter

7.1 myHDL Module

In [46]:

```
#Create the Direction States for Johnson Counter
DirStates=enum('Left','Halt','Right')
print(f"`Left` state repersentation is {bin(DirStates.Left)}")
print(f"`Halt` state repersentation is {bin(DirStates.Halt)}")
print(f"`Right` state repersentation is {bin(DirStates.Right)}")
```

```
`Left` state repersentation is 0
`Halt` state repersentation is 1
`Right` state repersentation is 10
```

In [47]:

```
@block
def JohnsonCount3(Dir, q, clk, rst):
    """
    Based of the `jc2` exsample from the myHDL website
    http://www.myhdl.org/docs/examples/jc2.html

    Input:
        Dir(state): Left,Right, Halt Direction States
        clk(bool): input clock
        rst(bool): reset signal

    Ouput:
        q(bitVec): the values in the D flip flops(aka counter)
    """

    q_i=Signal(intbv(0)[len(q):])
    @always(clk.posedge, rst.negedge)
    def JCStateMachine():
        #moore state machine
        if rst:
            q_i.next=0

        elif Dir==DirStates.Left:
            #set bit slice from left most to one from the right
            #from bit slice from one to the left to the right most
            q_i.next[len(q_i):1]=q_i[len(q_i)-1:]
            #set next right most bit to negated one to the left bit
            q_i.next[0]=not q_i[len(q_i)-1]

        elif Dir==DirStates.Halt:
            #create circular stop
            q_i.next=q_i

        elif Dir==DirStates.Right:
            #set bit slice from one from the left to right most
            #from bit slice left most bit to one from the right
            q_i.next[len(q_i)-1:]=q_i[len(q_i):1]
            #set next one bit from the right to be negated left most bi
            q_i.next[len(q_i)-1]=not q_i[0]



    @always_comb
    def OuputBuffer():
        q.next=q_i

    return instances()
```

7.2 myHDL Testing

In [48]:

```
BitSize=4
Peeker.clear()
clk=Signal(bool(0)); Peeker(clk, 'clk')
rst=Signal(bool(0)); Peeker(rst, 'rst')
q=Signal(intbv(0)[BitSize:]); Peeker(q, 'q')
Dir=Signal(DirStates.Right); Peeker(Dir, 'Dir')

DUT=JohnsonCount3(Dir, q, clk, rst)

def JohnsonCount3_TB():
    """
    myHDL only Testbench for `RingCounter` module
    """
    @always(delay(1))
    def ClkGen():
        clk.next=not clk

    @instance
    def stimules():
        i=0
        while True:
            if i==2*2*BitSize:
                Dir.next=DirStates.Left
            elif i==4*2*BitSize:
                rst.next=1
            elif i==4*2*BitSize+1:
                rst.next=0
            elif i==4*2*BitSize+2:
                Dir.next=DirStates.Halt


            if i==5*2*BitSize:
                raise StopSimulation()

            i+=1
            yield clk.posedge

    return instances()

sim=Simulation(DUT, JohnsonCount3_TB(), *Peeker.instances()).run()
```

In [49]:

```
Peeker.to_wavedrom()
```

In [50]:
```
JohnsonCount3Data=Peeker.to_dataframe()
JohnsonCount3Data=JohnsonCount3Data[JohnsonCount3Data['clk']==1]
JohnsonCount3Data.drop('clk', axis=1, inplace=True)
JohnsonCount3Data.reset_index(drop=True, inplace=True)
JohnsonCount3Data
```

Out[50]:

	Dir	q	rst
0	Right	8	0
1	Right	12	0
2	Right	14	0
3	Right	15	0
4	Right	7	0
5	Right	3	0
6	Right	1	0
7	Right	0	0
8	Right	8	0
9	Right	12	0
10	Right	14	0
11	Right	15	0

In [51]:
```
JohnsonCount3Data['q']=JohnsonCount3Data['q'].apply(lambda x:bin(x, Bit
JohnsonCount3Data
```

Out[51]:

	Dir	q	rst
0	Right	1000	0
1	Right	1100	0
2	Right	1110	0
3	Right	1111	0
4	Right	0111	0
5	Right	0011	0
6	Right	0001	0
7	Right	0000	0
8	Right	1000	0
9	Right	1100	0
10	Right	1110	0
11	Right	1111	0

In [52]:
```
JohnsonCount3Data[JohnsonCount3Data['Dir']==DirStates.Right]
```

Out[52]:

	Dir	q	rst
0	Right	1000	0
1	Right	1100	0
2	Right	1110	0
3	Right	1111	0
4	Right	0111	0
5	Right	0011	0
6	Right	0001	0
7	Right	0000	0
8	Right	1000	0
9	Right	1100	0
10	Right	1110	0
11	Right	1111	0
12	Right	0111	0
13	Right	0011	0
14	Right	0001	0

In [53]:
```
JohnsonCount3Data[JohnsonCount3Data['Dir']==DirStates.Left]
```

Out[53]:

	Dir	q	rst
15	Left	0000	0
16	Left	0001	0
17	Left	0011	0
18	Left	0111	0
19	Left	1111	0
20	Left	1110	0
21	Left	1100	0
22	Left	1000	0
23	Left	0000	0
24	Left	0001	0
25	Left	0011	0
26	Left	0111	0
27	Left	1111	0
28	Left	1110	0
29	Left	1100	0
30	Left	1000	0
31	Left	0000	1
32	Left	0001	0

In [54]:
```
JohnsonCount3Data[JohnsonCount3Data['Dir']==DirStates.Halt]
```

Out[54]:

	Dir	q	rst
33	Halt	0011	0
34	Halt	0011	0
35	Halt	0011	0
36	Halt	0011	0
37	Halt	0011	0
38	Halt	0011	0

7.3 Verilog Code

In [55]:
```
DUT.convert()
VerilogTextReader('JohnsonCount3');
```

Verilog modual from JohnsonCount3.v

```
 // File: JohnsonCount3.v
// Generated by MyHDL 0.10
// Date: Wed Sep  5 07:53:01 2018


`timescale 1ns/10ps

module JohnsonCount3 (
    Dir,
    q,
    clk,
    rst
);
// Based of the `jc2` exsample from the myHDL website
// http://www.myhdl.org/docs/examples/jc2.html (http://www.myhdl.org/do
cs/examples/jc2.html)
//
// Input:
//     Dir(state): Left,Right, Halt Direction States
//     clk(bool): input clock
//     rst(bool): reset signal
//
// Ouput:
//     q(bitVec): the values in the D flip flops(aka counter)

input [1:0] Dir;
output [3:0] q;
wire [3:0] q;
input clk;
input rst;

reg [3:0] q_i = 0;



always @(posedge clk, negedge rst) begin: JOHNSONCOUNT3_JCSTATEMACHINE
    if (rst) begin
        q_i <= 0;
    end
    else if ((Dir == 2'b00)) begin
        q_i[4-1:1] <= q_i[(4 - 1)-1:0];
        q_i[0] <= (!q_i[(4 - 1)]);
    end
    else if ((Dir == 2'b01)) begin
        q_i <= q_i;
    end
    else if ((Dir == 2'b10)) begin
        q_i[(4 - 1)-1:0] <= q_i[4-1:1];
        q_i[(4 - 1)] <= (!q_i[0]);
    end
end
```

```
assign q = q_i;

endmodule
```

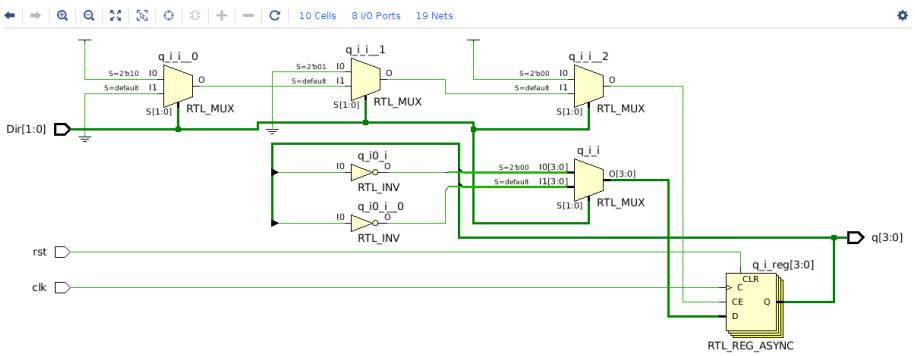

Figure 14: JohnsonCount3 RTL schematic; Xilinx Vivado 2017.4

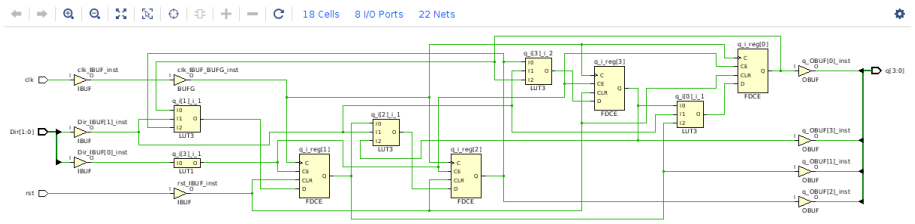

Figure 15: JohnsonCount3 Synthesized Schematic; Xilinx Vivado 2017.4

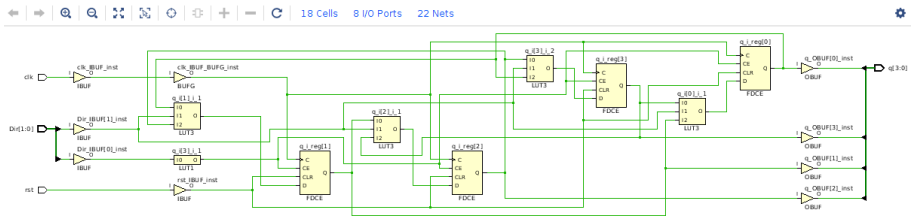

Figure 16: JohnsonCount3 Implementated Schematic; Xilinx Vivado 2017.4

7.4 Verilog Testbench

In [56]:

```
@block
def JohnsonCount3_TBV():
    """
    myHDL -> Verilog Testbench for `UpDown_Counter` module
    """

    clk=Signal(bool(0))
    rst=Signal(bool(0))
    q=Signal(intbv(0)[BitSize:])
    Dir=Signal(DirStates.Right)

    @always_comb
    def print_data():
        print(clk, rst, q, Dir)

    DUT=JohnsonCount3(Dir, q, clk, rst)

    @instance
    def clk_signal():
        while True:
            clk.next = not clk
            yield delay(1)

    @instance
    def stimules():
        i=0
        while True:
            if i==2*2*BitSize:
                Dir.next=DirStates.Left
            elif i==4*2*BitSize:
                rst.next=1
            elif i==4*2*BitSize+1:
                rst.next=0
            elif i==4*2*BitSize+2:
                Dir.next=DirStates.Halt
            else:
                pass


            if i==5*2*BitSize:
                raise StopSimulation()

            i+=1
            yield clk.posedge

    return instances()

TB=JohnsonCount3_TBV()
TB.convert(hdl="Verilog", initial_values=True)
VerilogTextReader('JohnsonCount3_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 modual from JohnsonCount3_TBV.v***
```

```
 // File: JohnsonCount3_TBV.v
// Generated by MyHDL 0.10
// Date: Wed Sep  5 07:53:06 2018


`timescale 1ns/10ps

module JohnsonCount3_TBV (

);
// myHDL -> Verilog Testbench for `UpDown_Counter` module


reg clk = 0;
reg rst = 0;
wire [3:0] q;
reg [1:0] Dir = 2'b10;
reg [3:0] JohnsonCount30_0_q_i = 0;



always @(rst, q, Dir, clk) begin: JOHNSONCOUNT3_TBV_PRINT_DATA
    $write("%h", clk);
    $write(" ");
    $write("%h", rst);
    $write(" ");
    $write("%h", q);
    $write(" ");
    $write("%h", Dir);
    $write("\n");
end


always @(posedge clk, negedge rst) begin: JOHNSONCOUNT3_TBV_JOHNSONCOUN
T30_0_JCSTATEMACHINE
    if (rst) begin
        JohnsonCount30_0_q_i <= 0;
    end
    else if ((Dir == 2'b00)) begin
        JohnsonCount30_0_q_i[4-1:1] <= JohnsonCount30_0_q_i[(4 - 1)-1:
0];
        JohnsonCount30_0_q_i[0] <= (!JohnsonCount30_0_q_i[(4 - 1)]);
    end
    else if ((Dir == 2'b01)) begin
        JohnsonCount30_0_q_i <= JohnsonCount30_0_q_i;
    end
    else if ((Dir == 2'b10)) begin
        JohnsonCount30_0_q_i[(4 - 1)-1:0] <= JohnsonCount30_0_q_i[4-1:
1];
        JohnsonCount30_0_q_i[(4 - 1)] <= (!JohnsonCount30_0_q_i[0]);
    end
end



assign q = JohnsonCount30_0_q_i;
```

```
initial begin: JOHNSONCOUNT3_TBV_CLK_SIGNAL
    while (1'b1) begin
        clk <= (!clk);
        # 1;
    end
end


initial begin: JOHNSONCOUNT3_TBV_STIMULES
    integer i;
    i = 0;
    while (1'b1) begin
        if ((i == ((2 * 2) * 4))) begin
            Dir <= 2'b00;
        end
        else if ((i == ((4 * 2) * 4))) begin
            rst <= 1;
        end
        else if ((i == (((4 * 2) * 4) + 1))) begin
            rst <= 0;
        end
        else if ((i == (((4 * 2) * 4) + 2))) begin
            Dir <= 2'b01;
        end
        else begin
            // pass
        end
        if ((i == ((5 * 2) * 4))) begin
            $finish;
        end
        i = i + 1;
        @(posedge clk);
    end
end

endmodule
```

7.5 PYNQ-Z1 Deployment

7.5.1 Board Constraints

In [57]:
```
ConstraintXDCTextReader('PYNQ_Z1Constraints_JohnsonCount3');
```

```
***Constraint file from PYNQ_Z1Constraints_JohnsonCount3.xdc***

 ## PYNQ-Z1 Constraint File for JohnsonCount3
## Based on https://github.com/Xilinx/PYNQ/blob/master/sdbuild/boot_con
figs/Pynq-Z1-defconfig/constraints.xdc (https://github.com/Xilinx/PYNQ/
blob/master/sdbuild/boot_configs/Pynq-Z1-defconfig/constraints.xdc)


##Switches

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

##LEDs

set_property -dict {PACKAGE_PIN R14 IOSTANDARD LVCMOS33} [get_ports {q
[0]}]; ##LED0
set_property -dict {PACKAGE_PIN P14 IOSTANDARD LVCMOS33} [get_ports {q
[1]}]; ##LED1
set_property -dict {PACKAGE_PIN N16 IOSTANDARD LVCMOS33} [get_ports {q
[2]}]; ##LED2
set_property -dict {PACKAGE_PIN M14 IOSTANDARD LVCMOS33} [get_ports {q
[3]}]; ##LED3

set_property -dict { PACKAGE_PIN D19   IOSTANDARD LVCMOS33 } [get_ports
 { rst }]; ##btn[0]
set_property -dict { PACKAGE_PIN L19   IOSTANDARD LVCMOS33 } [get_ports
 { clk }]; ##btn[3]
## Needed since if constraints even thinks a clock port is going to be
 connected to a non clock driver it wont synthize without it
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets {clk}];
##should only be done for teaching and realy only on LOW (nearly none)
 jitter (Bouncy) sources
```

Pay attention to line 22 that follows of setting the `clk` input signal to Button 3

```
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets {clk}];
```

This is needed in the constraint file in order for the Implementation and Bitstream to work. What this line says to Vivado is "I know that a clock signal is hooked up to a nonstandard clock source, go ahead and make the connection so". This is because the internal rule checking in vivado will raise errors if one attempts to connect a nonclock source to what it perceives (in this case correctly) should be a clock input. Normally this should not be done this way. But because this is for teaching and the `JohnsonCount3` was to be implemented as a stand-alone module without a clock divider at a clock speed of Hertz (FPGAs typically have Megahertz built-in clocks) this had to be done.

7.5.2 Deployment Results

YouTube:Bi-Directional Johnson Counter from myHDL on the PYNQ-Z1 (https://www.youtube.com/watch?v=UulvPq7Tk_Q)

8 Cyclic Shift Register Ring Counter

Like the Johnson Counter, which is the Mobius ring version of the Ring Counter, a Ring Counter is, in reality, a cyclic shift register since it simply shift the bits in its memory left or right. The following are aspects of Ring Counter by Sougata Bhattacharjee [https://www.quora.com/What-is-the-difference-between-a-Johnson-counter-and-a-ring-counter (https://www.quora.com/What-is-the-difference-between-a-Johnson-counter-and-a-ring-counter)]

- In a ring counter, the output of the last flip-flop is connected to the input of the first flip-flop.
- If n is the number of flip-flops that are used in ring counter, the number of possible states are also n.That means the number of states is equal to the number of flip-flops used.
- A Ring counter is mostly used in Successive approximation type ADC and stepper motor control.
- Decoding is easy in a ring counter as the number of states is equal to the number of flip-flops.
- If the input frequency to a ring counter is f then the output frequency $\frac{f}{n}$.
- The total number of unused states in the ring counter is $(2^n - n)$.

For a four-bit ring counter, the next state diagram is given by the following table from wikipedga (https://en.wikipedia.org/wiki/Ring_counter)

Straight ring counter				
State	Q0	Q1	Q2	Q3
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1
0	1	0	0	0

Figure 17: 4-bit Ring Counter next state table from wikipedia (https://en.wikipedia.org/wiki/Ring_counter#Four-bit_ring-counter_sequences)

8.1 myHDL Module

In [58]:
```
#Create the Direction States for Ring Counter
DirStates=enum('Left','Halt','Right')
print(f"`Left` state repersentation is {bin(DirStates.Left)}")
print(f"`Halt` state repersentation is {bin(DirStates.Halt)}")
print(f"`Right` state repersentation is {bin(DirStates.Right)}")
```

```
`Left` state repersentation is 0
`Halt` state repersentation is 1
`Right` state repersentation is 10
```

In [59]:
```
@block
def RingCounter(seed, Dir, q, clk, rst):
    """
    Seedable and direction controlable ring counter in myHDL

    Input:
        seed(bitvec): intial value for ring counter
        Dir(enum): Direction contorl signal
        clk(bool): clock
        rst(bool): reset
    Output
    """
    q_i=Signal(intbv(int(seed))[len(q):])
    @always(clk.posedge, rst.negedge)
    def RCStateMachine():
        #moore state machine
        if rst:
            q_i.next=seed
        elif Dir==DirStates.Left:
            q_i.next=concat(q_i[len(q_i)-1:0],q_i[len(q_i)-1])

        elif Dir==DirStates.Halt:
            #create circular stop
            q_i.next=q_i

        elif Dir==DirStates.Right:
            q_i.next=concat(q_i[0], q_i[len(q_i):1])

    @always_comb
    def OuputBuffer():
        q.next=q_i

    return instances()
```

8.2 myHDL testing

In [60]:

```
BitSize=4; seedval=3
Peeker.clear()
seed=Signal(intbv(seedval)[BitSize:]); Peeker(seed, 'seed')
clk=Signal(bool(0)); Peeker(clk, 'clk')
rst=Signal(bool(0)); Peeker(rst, 'rst')
q=Signal(intbv(0)[BitSize:]); Peeker(q, 'q')
Dir=Signal(DirStates.Right); Peeker(Dir, 'Dir')

DUT=RingCounter(seed, Dir, q, clk, rst)

def RingCounter_TB():
    """
    myHDL only Testbench for `RingCounter` module
    """
    @always(delay(1))
    def ClkGen():
        clk.next=not clk

    @instance
    def stimules():
        i=0
        while True:
            if i==2*BitSize:
                Dir.next=DirStates.Left
            elif i==3*BitSize:
                rst.next=1
            elif i==3*BitSize+1:
                rst.next=0
            elif i==3*BitSize+2:
                Dir.next=DirStates.Halt


            if i==5*BitSize:
                raise StopSimulation()

            i+=1
            yield clk.posedge

    return instances()

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

In [61]:

```
Peeker.to_wavedrom()
```

In [62]:

```
RingCountData=Peeker.to_dataframe()
RingCountData=RingCountData[RingCountData['clk']==1]
RingCountData.drop('clk', axis=1, inplace=True)
RingCountData.reset_index(drop=True, inplace=True)
RingCountData
```

Out[62]:

	Dir	q	rst	seed
0	Right	9	0	3
1	Right	12	0	3
2	Right	6	0	3
3	Right	3	0	3
4	Right	9	0	3
5	Right	12	0	3
6	Right	6	0	3
7	Left	3	0	3
8	Left	6	0	3
9	Left	12	0	3
10	Left	9	0	3
11	Left	3	1	3

In [63]:
```
RingCountData['q']=RingCountData['q'].apply(lambda x:bin(x, BitSize))
RingCountData
```

Out[63]:

	Dir	q	rst	seed
0	Right	1001	0	3
1	Right	1100	0	3
2	Right	0110	0	3
3	Right	0011	0	3
4	Right	1001	0	3
5	Right	1100	0	3
6	Right	0110	0	3
7	Left	0011	0	3
8	Left	0110	0	3
9	Left	1100	0	3
10	Left	1001	0	3
11	Left	0011	1	3
12	Left	0110	0	3
13	Halt	1100	0	3
14	Halt	1100	0	3
15	Halt	1100	0	3
16	Halt	1100	0	3
17	Halt	1100	0	3
18	Halt	1100	0	3

In [64]:
```
RingCountData[RingCountData['Dir']==DirStates.Right]
```

Out[64]:

	Dir	q	rst	seed
0	Right	1001	0	3
1	Right	1100	0	3
2	Right	0110	0	3
3	Right	0011	0	3
4	Right	1001	0	3
5	Right	1100	0	3
6	Right	0110	0	3

In [65]:
```
RingCountData[RingCountData['Dir']==DirStates.Left]
```

Out[65]:

	Dir	q	rst	seed
7	Left	0011	0	3
8	Left	0110	0	3
9	Left	1100	0	3
10	Left	1001	0	3
11	Left	0011	1	3
12	Left	0110	0	3

In [66]:
```
RingCountData[RingCountData['Dir']==DirStates.Halt]
```

Out[66]:

	Dir	q	rst	seed
13	Halt	1100	0	3
14	Halt	1100	0	3
15	Halt	1100	0	3
16	Halt	1100	0	3
17	Halt	1100	0	3
18	Halt	1100	0	3

8.3 Verilog Code

In [67]:
```
DUT.convert()
VerilogTextReader('RingCounter');
```

Verilog modual from RingCounter.v

```
 // File: RingCounter.v
// Generated by MyHDL 0.10
// Date: Wed Sep  5 07:53:27 2018


`timescale 1ns/10ps

module RingCounter (
    seed,
    Dir,
    q,
    clk,
    rst
);
// Seedable and direction controlable ring counter in myHDL
//
// Input:
//     seed(bitvec): intial value for ring counter
//     Dir(enum): Direction contorl signal
//     clk(bool): clock
//     rst(bool): reset
// Output

input [3:0] seed;
input [1:0] Dir;
output [3:0] q;
wire [3:0] q;
input clk;
input rst;

reg [3:0] q_i = 3;



always @(posedge clk, negedge rst) begin: RINGCOUNTER_RCSTATEMACHINE
    if (rst) begin
        q_i <= seed;
    end
    else if ((Dir == 2'b00)) begin
        q_i <= {q_i[(4 - 1)-1:0], q_i[(4 - 1)]};
    end
    else if ((Dir == 2'b01)) begin
        q_i <= q_i;
    end
    else if ((Dir == 2'b10)) begin
        q_i <= {q_i[0], q_i[4-1:1]};
    end
end



assign q = q_i;
```

```
endmodule
```

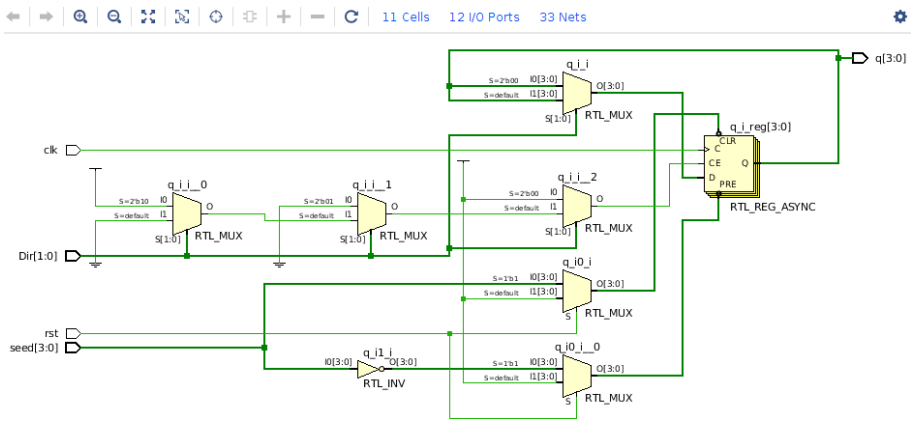

Figure 18: RingCounter RTL schematic; Xilinx Vivado 2017.4

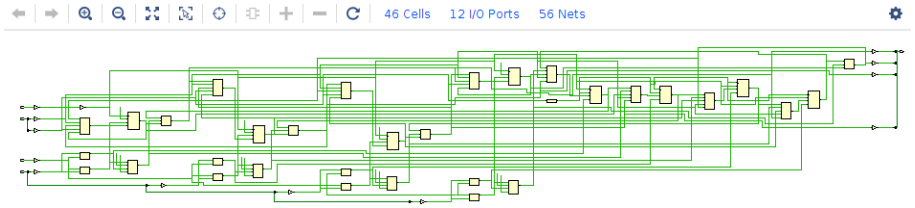

Figure 19: RingCounter Synthesized Schematic; Xilinx Vivado 2017.4

8.4 Verilog Testbench

In [68]:

```
BitSize=4; seedval=3
@block
def RingCounter_TBV():
    """
    myHDL -> verilog Testbench for `RingCounter` module
    """
    seed=Signal(intbv(seedval)[BitSize:])
    clk=Signal(bool(0))
    rst=Signal(bool(0))
    q=Signal(intbv(0)[BitSize:])
    Dir=Signal(DirStates.Right)

    @always_comb
    def print_data():
        print(seed, Dir, q, clk, rst)

    DUT=RingCounter(seed, Dir, q, clk, rst)

    @instance
    def clk_signal():
        while True:
            clk.next = not clk
            yield delay(1)

    @instance
    def stimules():
        i=0
        while True:
            if i==2*BitSize:
                Dir.next=DirStates.Left
            elif i==3*BitSize:
                rst.next=1
            elif i==3*BitSize+1:
                rst.next=0
            elif i==3*BitSize+2:
                Dir.next=DirStates.Halt


            if i==5*BitSize:
                raise StopSimulation()

            i+=1
            yield clk.posedge

    return instances()



TB=RingCounter_TBV()
TB.convert(hdl="Verilog", initial_values=True)
VerilogTextReader('RingCounter_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'>
```

Verilog modual from RingCounter_TBV.v

```
 // File: RingCounter_TBV.v
// Generated by MyHDL 0.10
// Date: Wed Sep  5 07:53:29 2018


`timescale 1ns/10ps

module RingCounter_TBV (

);
// myHDL -> verilog Testbench for `RingCounter` module


reg clk = 0;
reg rst = 0;
wire [3:0] q;
reg [1:0] Dir = 2'b10;
wire [3:0] seed;
reg [3:0] RingCounter0_0_q_i = 3;

assign seed = 4'd3;


always @(q, rst, Dir, clk, seed) begin: RINGCOUNTER_TBV_PRINT_DATA
    $write("%h", seed);
    $write(" ");
    $write("%h", Dir);
    $write(" ");
    $write("%h", q);
    $write(" ");
    $write("%h", clk);
    $write(" ");
    $write("%h", rst);
    $write("\n");
end


always @(posedge clk, negedge rst) begin: RINGCOUNTER_TBV_RINGCOUNTER0_
0_RCSTATEMACHINE
    if (rst) begin
        RingCounter0_0_q_i <= seed;
    end
    else if ((Dir == 2'b00)) begin
        RingCounter0_0_q_i <= {RingCounter0_0_q_i[(4 - 1)-1:0], RingCou
nter0_0_q_i[(4 - 1)]};
    end
    else if ((Dir == 2'b01)) begin
        RingCounter0_0_q_i <= RingCounter0_0_q_i;
    end
    else if ((Dir == 2'b10)) begin
        RingCounter0_0_q_i <= {RingCounter0_0_q_i[0], RingCounter0_0_q_
i[4-1:1]};
    end
end
```

```
assign q = RingCounter0_0_q_i;


initial begin: RINGCOUNTER_TBV_CLK_SIGNAL
    while (1'b1) begin
        clk <= (!clk);
        # 1;
    end
end


initial begin: RINGCOUNTER_TBV_STIMULES
    integer i;
    i = 0;
    while (1'b1) begin
        if ((i == (2 * 4))) begin
            Dir <= 2'b00;
        end
        else if ((i == (3 * 4))) begin
            rst <= 1;
        end
        else if ((i == ((3 * 4) + 1))) begin
            rst <= 0;
        end
        else if ((i == ((3 * 4) + 2))) begin
            Dir <= 2'b01;
        end
        if ((i == (5 * 4))) begin
            $finish;
        end
        i = i + 1;
        @(posedge clk);
    end
end

endmodule


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

8.5 PYNQ-Z1 Deployment

8.5.1 Block Design

Video on how Block Design was made found here: YouTube:Seeded Ring Counter RTL IP Hookup in Vivado from myHDL to PYNQ-Z1 (https://youtu.be/vnCKs0hQq3U)

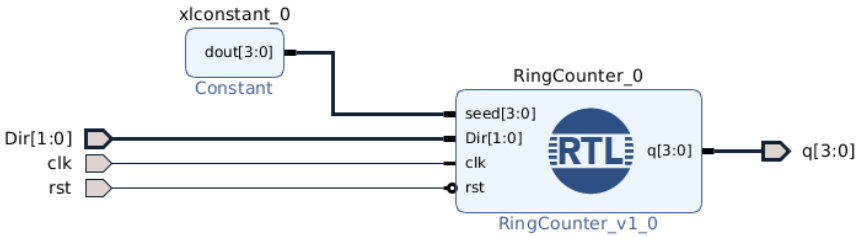

Figure 20: RingCounter Block IP Block Design; Xilinx Vivado 2017.4

The Constant IP in the top left corner of the Block Design has the following internal parameterizations, accessed by right clicking on the IP and selecting "Customize Block"

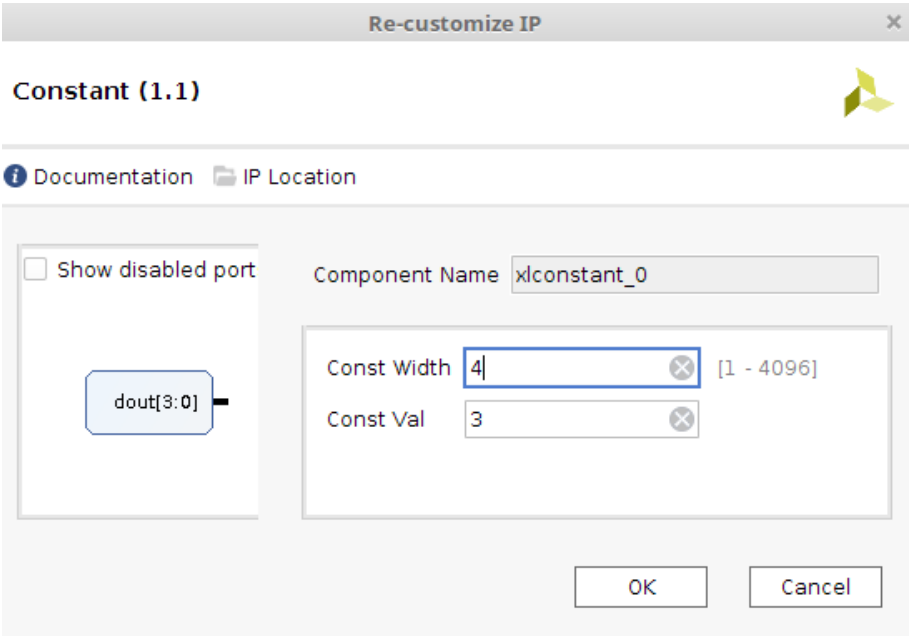

Figure 21: Xilinx IP Constant 1.1 Internal Settings for Ring Counter Block design ; Xilinx Vivado 2017.4

8.5.2 Board Constraints

In [69]:
```
ConstraintXDCTextReader('PYNQ_Z1Constraints_RingCounterBlock');
```

```
***Constraint file from PYNQ_Z1Constraints_RingCounterBlock.xdc***

 ## PYNQ-Z1 Constraint File for RingCounterBlock
## Based on https://github.com/Xilinx/PYNQ/blob/master/sdbuild/boot_con
figs/Pynq-Z1-defconfig/constraints.xdc (https://github.com/Xilinx/PYNQ/
blob/master/sdbuild/boot_configs/Pynq-Z1-defconfig/constraints.xdc)


##Switches

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

##LEDs

set_property -dict {PACKAGE_PIN R14 IOSTANDARD LVCMOS33} [get_ports {q
[0]}]; ##LED0
set_property -dict {PACKAGE_PIN P14 IOSTANDARD LVCMOS33} [get_ports {q
[1]}]; ##LED1
set_property -dict {PACKAGE_PIN N16 IOSTANDARD LVCMOS33} [get_ports {q
[2]}]; ##LED2
set_property -dict {PACKAGE_PIN M14 IOSTANDARD LVCMOS33} [get_ports {q
[3]}]; ##LED3

set_property -dict { PACKAGE_PIN D19   IOSTANDARD LVCMOS33 } [get_ports
 { rst }]; ##btn[0]
set_property -dict { PACKAGE_PIN L19   IOSTANDARD LVCMOS33 } [get_ports
 { clk }]; ##btn[3]
## Needed since if constraints even thinks a clock port is going to be
 connected to a non clock driver it wont synthize without it
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets {clk}];
##should only be done for teaching and realy only on LOW (nearly none)
 jitter (Bouncy) sources
```

8.5.3 RTL, Synthesis, & Implementation

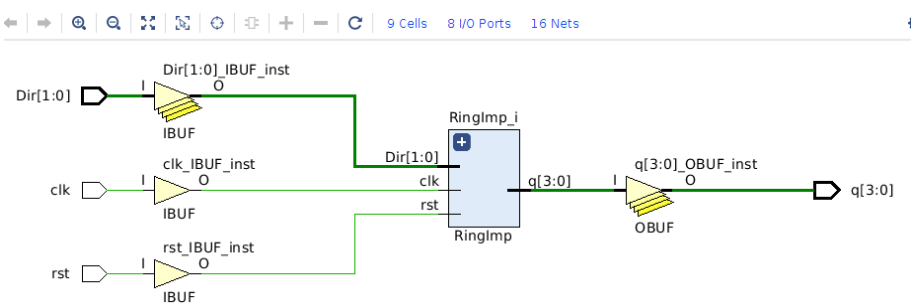

Figure 22: Ring Counter Block RTL schematic Level 1; Xilinx Vivado 2017.4

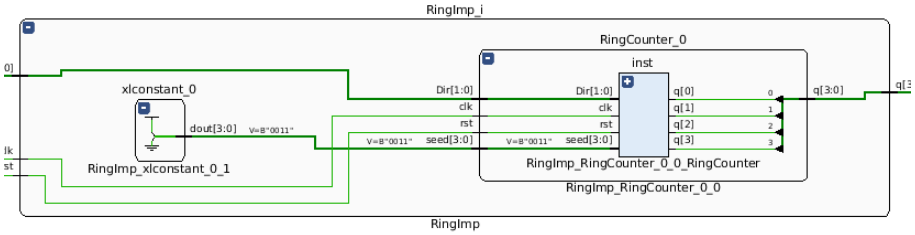

Figure 23: Ring Counter Block RTL `RingImp_i` schematic internal; Xilinx Vivado 2017.4

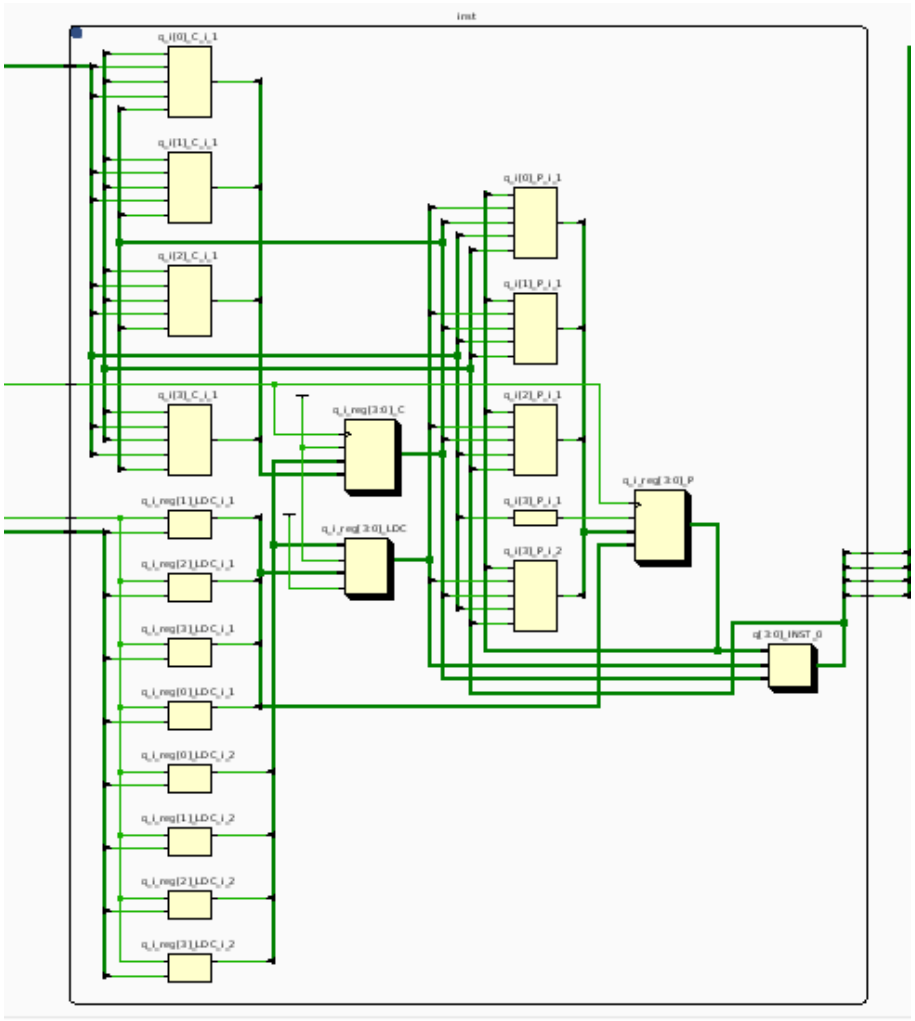

Figure 24: Ring Counter Block RTL `RingCounter_0` `inst` schematic internal; Xilinx Vivado 2017.4

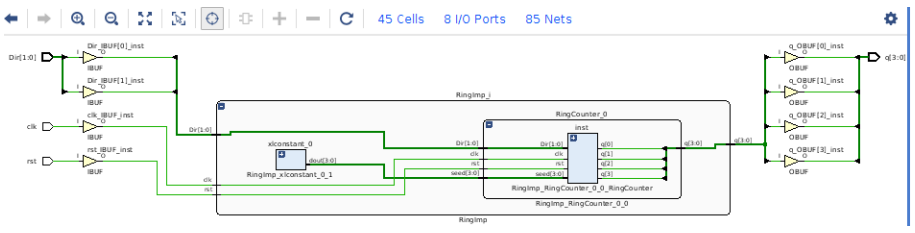

Figure 25: Ring Counter Block Synthesized Schematic; Xilinx Vivado 2017.4

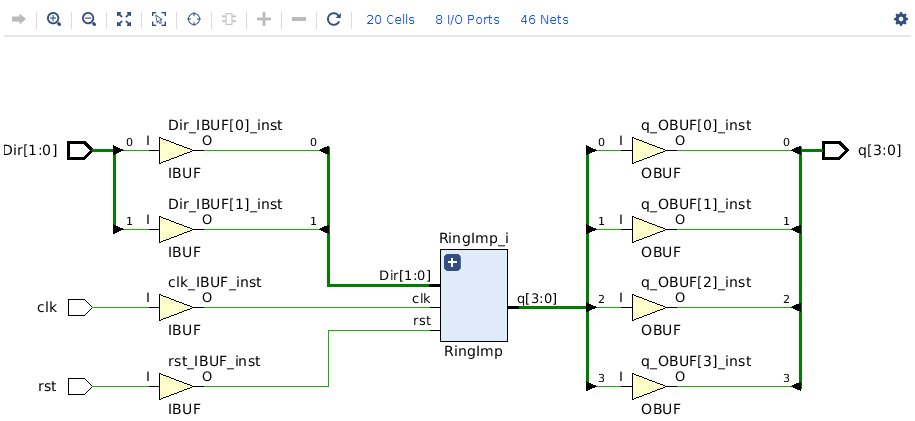

Figure 26: Ring Counter Block Implementated Schematic Top Level; Xilinx Vivado 2017.4

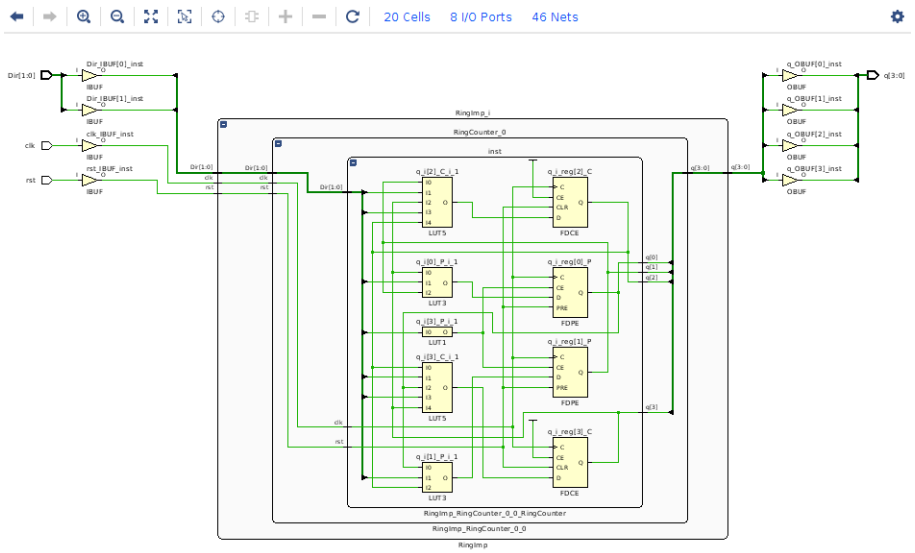

Figure 27: Ring Counter Block Implementated Schematic Expanded; Xilinx Vivado 2017.4

8.5.4 Deployment Results

YouTube:[Seeded Ring Counter from myHDL on PYNQ-Z1 (https://www.youtube.com/watch?v=7ZQ4qCvjokU)](https://www.youtube.com/watch?v=7ZQ4qCvjokU)