**PROBLEM STATEMENT**

This experiment observes the application of parameter redefinitions by the user to create multiplexing circuits of variable size. In addition, to handle bits not defined by the user, multiple methods to discern undefined bits during selector circuit operation are utilized.

**KEYWORDS:** MUX, Logical High, Logical Low, Undefined, High Impedance, Hi-Z, 1, 0, X, Z, Select Line, Control Signal.

1. **INTRODUCTION**

The process of multiplexing (MUX) consists of selecting one or more inputs based on a control signal or select line. The simplest multiplexer is one that acts as a single-pole double-throw switch, selecting which of two inputs are passed to the output on the basis of a third control input. In digital design, multiplexing is used extensively to implement conditional logic.

The simple multiplexing circuit form Experiment #1 Familiarization with Linux and the Synopsys VCS Simulator is utilized in this experiment to provide contrast to the behavior of the scaling mux. The MUX2\_1 module is described below in Figure 5.1. The truth table for this circuit follows in table 5.1

*Figure 5.1* Gate Schematic for 2-Input MUX

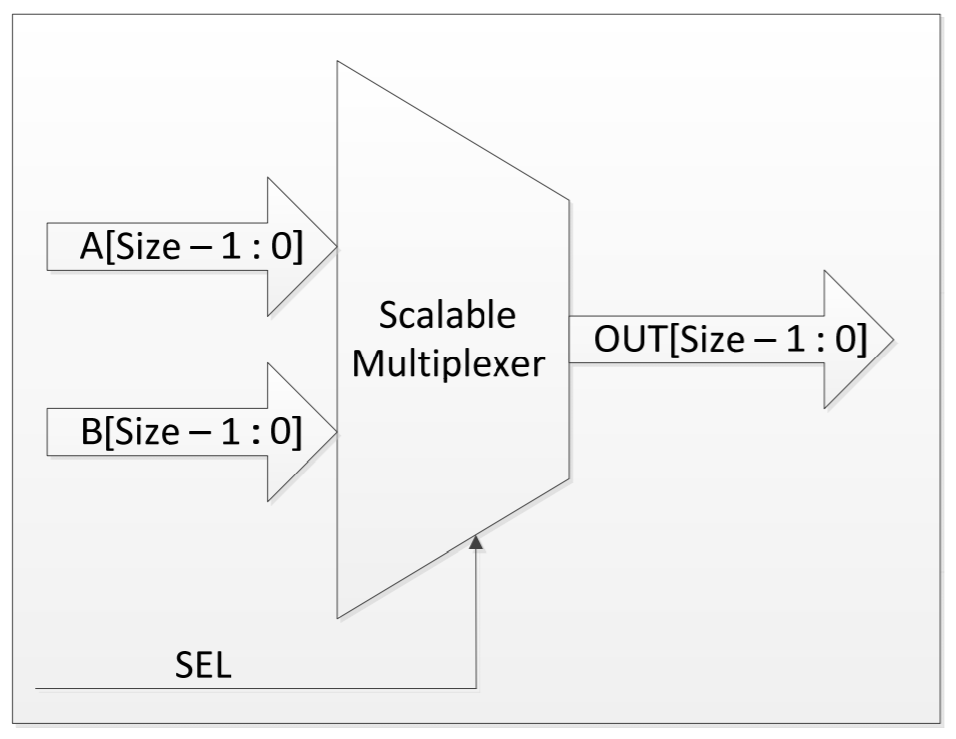


*Table 5.1* Single BitMUX2\_1 Truth Table

|  |  |  |  |
| --- | --- | --- | --- |
| *A* | *B* | *SEL* | OUT |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

In this simple multiplexing circuit, only a 1-bit input line is considered. To multiplex between two different input busses, similar logic can be used, however there are multiple design considerations regarding the bus width which must be taken into account. In this experiment, the bus width in the synthesized circuit varies based on the provided data from the user. A high level view of the scalable multiplexer circuit is shown below in Figure 5.2.

*Figure 5.2* scale\_mux High Level Diagram



The multiplexer selects between two inputs of Y bits of length; implying the corresponding output determined by the Select Control Signal and must also be Y bits wide. The truth table of the Scalable Multiplexer above is shown below in Table 5.2. Just as with the simple multiplexer above, the select line can assume logical low and logical high values. However, this experiment also considers conditions where the select line is ambiguous. A` and B` represent unsigned binary values which range from 0 to 2Y-1 where A` does not equal B`.

*Table 5.2* Scale\_Mux Truth Table

|  |  |  |  |
| --- | --- | --- | --- |
| *A[Y:0]* | *B[Y:0]* | *SEL* | OUT*[Y:0]* |
| A` | B` | 0 | A` |
| A` | B` | 1 | B` |
| A` | A` | X | A` |
| A` | B` | X | X |

1. **METHODOLOGY**

To determine the behavior of the experimental scalable multiplexer circuit, multiple input widths must be tested. IN addition, the behavior of the scalable multiplexing circuit must be contrasted with the fundamental multiplexer, MUX2\_1 above, to correct behavior when no width is defined.

To satisfy this collection of data, two Verilog modules are constructed. One module contains the declaration and behavioral definition of the scalable multiplexer circuit using SystemVerilog code. This module must be able to handle all combinational inputs of the binary set {1,0,X} and output a bus of varying width controlled by the Select (*SEL*) input. The width is defined by the SIZE parameter. This module can be seen below in section 5.3 as Module 5.1 *Scale\_mux.sv*. In addition to the scalable multiplexer module, a testbench module must also be implemented to observe the *Scale\_mux* module’s output given different input conditions and widths. This test bench module instantiates four scalable multiplexer circuits of varying widths, arbitrarily selected from the author’s numeric identification string, tied to the same bus inputs. A one bit wide multiplexer is included in the default width case. This module is also detailed in section 5.3 and can be seen as Module 5.2 *tb\_ Scale\_Mux.v*.

The input conditions required to test the scalable multiplexing circuit are considered and test vectors are chosen accordingly. Four distinct cases, one per scalable multiplexer instantiation, are defined: When select is known and is Logical High, when select is known and is Logical Low, when select is unknown and both inputs are equal and finally when select is unknown and the inputs are not equal. In the last case described, All instances of the scalable multiplexer are simulated at the same time. Cases where inputs are of differing widths are not considered.

The testbench and behavioral modules must then be compiled using the Verilog Compile Simulator tool (VCS). If compiled with no warnings or errors, the behavioral simulation will be run and the output recorded. Table 5.3 shows the tabulated results captured during simulation.

1. **MODULE FILES & SIMULATION RESULTS**

*Module 5.1*—scale\_mux.sv

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\* \*\*\*

\*\*\* EE 526 L Experiment #5 Kyle E. Keislar, Spring, 2020 \*\*\*

\*\*\* \*\*\*

\*\*\* Scalable Multiplexer \*\*\*

\*\*\* \*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\* Filename: scale\_mux.v \*\*\*

\*\*\* Author: Kyle E. Keislar \*\*\*

\*\*\* Date: 02/27/2020 \*\*\*

\*\*\* Version: 1.0 \*\*\*

\*\*\* Revised: \*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\* Module Description: scalable 2, input multiplexer which can handle data

bus inputs of user defined width \*\*\*

\*\*\* \*\*\*

\*\*\* \*\*\*

\*\*\* \*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

`timescale 1ns **/** 10ps

// Module Declaration

**module** scale\_mux**(** A**,** //First data input of the Multiplexer; can be SIZE bits wide and assume values from 0-2^SIZE-1

B**,** //Second data input of the Multiplexer; can be SIZE bits wide and assume values from 0-2^SIZE-1

SEL**,** //Multiplexer Select Line; 0 to select A, 1 to select B; may be indeterminate

OUT**);** //Multiplexer output bus; must be SIZE bits wide and assume values from 0-2^SIZE-1 to handle A or B inputs; selected by SEL

//Module Parameters

//Widths selected from ID: 3, 4, 5, 1(deafault)

**parameter** SIZE**=**1**;** //set default

// I/O port assignment

**output** **reg** **[**SIZE**-**1**:**0**]** OUT**;**

**input** **wire** **[**SIZE**-**1**:**0**]** A**,**B**;**

**input** **wire** SEL**;**

**integer** i**;**

//Internal Signals

//reg [SIZE-1:0] out\_reg;//wire from enable logic to output registers,

//Mux Logic

**always\_comb** **begin**

**if(**SEL**&&**1'B1**)** OUT**=**B**;** //if select is 1 chooose B

**else** **if(!(**SEL**||**1'B0**))** OUT**=**A**;** //if it is A

**else** **begin** //If sel is undetermineed

**for(**i**=**0**;**i**<**SIZE**;**i**=**i**+**1**)** **begin**

**if(**A**[**i**]^**B**[**i**])** OUT**[**i**]=**1'bX**;** //output conflicts if the bits are not the same

**else** OUT**[**i**]=**A**[**i**];** //if they are the same then give either a or b

**end**

**end**

**end**

**endmodule**

*Module 5.2*—tb\_scale\_mux.v

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\* \*\*\*

\*\*\* EE 526 L Experiment #5 Kyle E. Keislar, Spring, 2020 \*\*\*

\*\*\* \*\*\*

\*\*\* Scalable Multiplexer \*\*\*

\*\*\* \*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\* Filename: tb\_scale\_mux.v \*\*\*

\*\*\* Author: Kyle E. Keislar \*\*\*

\*\*\* Date: 02/20/2020 \*\*\*

\*\*\* Version: 1.0 \*\*\*

\*\*\* Revised: \*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\* Test Modules: scale\_mux.v \*\*\*

\*\*\* & Hierarchy : \*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\* Test Strategy: The scalable multiplexer is tested with synthesizable

methods by defining 4 different instances and parameters

to test values of various widths.

Instance Bit Width

1 5

2 3

3 4

4 1(default)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//Timescale

`timescale 1ns **/** 10ps

`define runtime 100**;**

// Module Instantiation

**module** tb\_scale\_mux**();**

//Module Parameters

//Widths selected from ID: 3, 4, 5, 1(deafault)

//Parameter definition

**parameter** SIZE**=**3**;**

//Parameter redfinition using defparam

//reg inputs

**reg** **[**4**:**0**]** a**,**b**;** //a,b inputs 5 bits wide for first instance

**reg** sel**;**

//wire outputs

**wire** **[**4**:**0**]** out1**,**out2**,**out3**,**out4**;**//output wire 5 bits wide for first instance

//Scalable Mux Instance 1: 5-bits wide

scale\_mux **#(**SIZE**)** UUT\_SM\_1**(**a**,**b**,**sel**,**out1**);**

**defparam** UUT\_SM\_1**.**SIZE**=**5**;**

//Parameter redifinition using hierarchal definition

//Scalable Mux Instance 2: 3-bits wide

scale\_mux **#(**SIZE**)** UUT\_SM\_2**(**a**[**2**:**0**],**b**[**2**:**0**],**sel**,**out2**[**2**:**0**]);**

//Parameter redfinition 1 using scoping operator

//Scalable Mux Instance 3: 4-bits wide

scale\_mux **#(.**SIZE**(**4**))** UUT\_SM\_3**(**a**[**3**:**0**],**b**[**3**:**0**],**sel**,**out3**[**3**:**0**]);**

//Parameter default definition

//Scalable Mux Instance 4: 1-bit wide

scale\_mux UUT\_SM\_4**(**a**[**0**:**0**],**b**[**0**:**0**],**sel**,**out4**[**0**:**0**]);**

//Initial Conditions

**initial** **begin**

$vcdpluson**;** //include waveforms in simulation

//initilize selects

sel**=**1'b0**;**

a**=**5'b10101**;** b**=**5'b01010**;**

**#**0 $display**(**"--------------------------------Initial conditions--------------------------------"**);**

**#**0 $display**(**"\t\t\ta1=%.5b\tb1=%.5b\tsel=%.5b"**,**a**,**b**,**sel**);**

**#**0 $display**(**"\t\t\tout1=%.5b\tout2=%.5b\tout3=%.5b\tout4=%.5b"**,**out1**,**out2**,**out3**,**out4**);**

**#**0 $display**(**"----------------------------------------------------------------------------------"**);**

$display**(**"sel==0"**);**

//test sel=0;

sel**=**1'b0**;**

**#**0 $display**(**"\t\t\ta1=%.5b\tb1=%.5b\tsel=%.5b"**,**a**,**b**,**sel**);**

**#**0 $display**(**"\t\t\tout1=%.5b\tout2=%.5b\tout3=%.5b\tout4=%.5b"**,**out1**,**out2**,**out3**,**out4**);**

$display**(**"sel==1"**);**

//test sel=1;

sel**=**1'b1**;**

**#**0 $display**(**"\t\t\ta1=%.5b\tb1=%.5b\tsel=%.5b"**,**a**,**b**,**sel**);**

**#**0 $display**(**"\t\t\tout1=%.5b\tout2=%.5b\tout3=%.5b\tout4=%.5b"**,**out1**,**out2**,**out3**,**out4**);**

$display**(**"sel==X, A==B"**);**

//test sel=X;

sel**=**1'bX**;**

//test A=B

a**=**15**;** b**=**15**;**

**#**0 $display**(**"\t\t\ta1=%.5b\tb1=%.5b\tsel=%.5b"**,**a**,**b**,**sel**);**

**#**0 $display**(**"\t\t\tout1=%.5b\tout2=%.5b\tout3=%.5b\tout4=%.5b"**,**out1**,**out2**,**out3**,**out4**);**

$display**(**"sel==X, A!=B"**);**

//test A!=B;

a**=**9**;** b**=**6**;**

**#**0 $display**(**"\t\t\ta1=%.5b\tb1=%.5b\tsel=%.5b"**,**a**,**b**,**sel**);**

**#**0 $display**(**"\t\t\tout1=%.5b\tout2=%.5b\tout3=%.5b\tout4=%.5b"**,**out1**,**out2**,**out3**,**out4**);**

//test A!=B;

a**=**25**;** b**=**19**;**

**#**0 $display**(**"\t\t\ta1=%.5b\tb1=%.5b\tsel=%.5b"**,**a**,**b**,**sel**);**

**#**0 $display**(**"\t\t\tout1=%.5b\tout2=%.5b\tout3=%.5b\tout4=%.5b"**,**out1**,**out2**,**out3**,**out4**);**

//Initilization (reset to 0)

**end**

//Asynchronous Behavior

//Synchronous Behavior

//Check starting values & Finish the simulation at runtime

**always**

**begin**

**#**`runtime

$finish**;**

**end**

**endmodule**

*Table 5.3* Tabulated Simulation Output for Scalable Multiplexer

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Condition** | **Inputs** | | **Outputs** | | | |
| *A* | *B* | *5 bits* | *3 bits* | *4 bits* | *1 bit* |
| Select is 0 | 10101 | 01010 | 10101 | ZZ101 | Z0101 | ZZZZ1 |
| Select is 1 | 10101 | 01010 | 01010 | ZZ010 | Z1010 | ZZZZ0 |
| Select is Unknown, A & B are equal | 01111 | 01111 | 01111 | ZZ111 | Z1111 | ZZZZ1 |
| Select is Unknown, A & B are not equal | 01001 | 00110 | 0XXXX | ZZXXX | ZXXXX | ZZZZX |
| Select is Unknown, A & B are not equal | 11001 | 10011 | 1X0X1 | ZZ0X1 | ZX0X1 | ZZZZ1 |

1. **ANALYSIS**

From the simulation outputs represented in Table 5.3 multiple conclusions about the functionality of the scalable multiplexer circuit can be derived. A summary of these observations is provided below. When referenced, the output of the MUX2\_1 module form experiment #1 can be viewed in the appendix as Item 5.7.

The scalable multiplexer circuit appears to function as a basic multiplexer when the SEL, A and B inputs are defined. However the simple and scalable multiplexers differs in their behavior when the select control signal is undefined. The scalable multiplexer can mitigate bit conflicts between the two inputs if both inputs are the same value, while the simple multiplexer will always output an unknown value when select is not defined. This discrepancy is due to the inherit behavior of the primitive gates and the way in which the two multiplexers are modeled. The simple multiplexer uses different primitive gates to structurally model a multiplexer and this results in the propagation of unknown values through the circuit resulting in an unknown output when the select line is not defined. However the scalable multiplexer is modeled behaviorally and uses combinational logic to assign values to the output.

In all other cases besides the default width, the circuit also behaves as a selector circuit. When the select line is low, the A input is passed to the output for all widths. When the select line is high, the B input is passed. When the select input is not defined, the scalable multiplexer attempts resolve any bit conflicts that arise from ambiguity between the two inputs. The module performs this function by assigning a single bit line with either ‘1’ or ‘0’ if both of the inputs possess the same value for the bit. However if the conflicting bit has a different value between A and B, the output bit cannot be determined and must remain unknown. This can be seen in the fourth and fifth rows of Table 5.3 as different values of A and B are asserted with varying bits sharing the same logical values. When the bits are the same the corresponding output bit is always defined, however when the bits are different the corresponding output is never defined.

Bits which are not defined by the user default to a high impedance state as examined from viewing the maximum width case of 5 bits and the default case of 1 bit.

1. **CONCLUSION**

From this experiment multiple conclusions about the behavior of the scalable multiplexing circuit. This experiment illustrated how undefined or values can be mitigated with behavioral descriptions as opposed to clunky structural ones. When the select value was not defined in the smiple multiplexer, the circuit output was not reliable, however using combinational logic the scalable multiplexer can resolve bit conflicts and partially handle some undefined input cases. However, both circuits only operate as multiplexing circuits when all control signals are defined.

This fundamental difference between the two circuits is deliberate in their design. The scalable multiplexer must have its behavior defined when an input is undefined, where as the simple multiplexer which uses primitive gates will default to a single case as shown in Experiment #1. This strength of behavioral description allows for more flexibility in design as well as being easier to represent in Verilog.



**APPENDIX**

1. **SIMULATION LOG A**

Chronologic VCS simulator copyright 1991-2017

Contains Synopsys proprietary information.

Compiler version N-2017.12-SP2-2\_Full64; Runtime version N-2017.12-SP2-2\_Full64; Mar 12 19:50 2020

VCD+ Writer N-2017.12-SP2-2\_Full64 Copyright (c) 19

91-2017 by Synopsys Inc.

--------------------------------Initial conditions--------------------------------

a1=10101 b1=01010 sel=0

out1=10101 out2=zz101 out3=z0101 out4=zzzz1

----------------------------------------------------------------------------------

sel==0

a1=10101 b1=01010 sel=0

out1=10101 out2=zz101 out3=z0101 out4=zzzz1

sel==1

a1=10101 b1=01010 sel=1

out1=01010 out2=zz010 out3=z1010 out4=zzzz0

sel==X, A==B

a1=01111 b1=01111 sel=x

out1=01111 out2=zz111 out3=z1111 out4=zzzz1

sel==X, A!=B

a1=01001 b1=00110 sel=x

out1=0xxxx out2=zzxxx out3=zxxxx out4=zzzzx

a1=11001 b1=10011 sel=x

out1=1x0x1 out2=zz0x1 out3=zx0x1 out4=zzzz1

$finish called from file "tb\_scale\_mux.v", line 127.

$finish at simulation time 10000

V C S S i m u l a t i o n R e p o r t

Time: 100000 ps

CPU Time: 0.250 seconds; Data structure size: 0.0Mb

Thu Mar 12 19:50:18 2020

1. **EXPERIMENT #1 SIMULATION OUTPUT : MUX2\_1**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Vector #** | **A** | **B** | **SEL** | **OUT** |
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 2 | 0 | 0 | Z | 0 |
| 3 | 0 | 0 | X | 0 |
| 4 | 0 | 1 | 0 | 0 |
| 5 | 0 | 1 | 1 | 1 |
| 6 | 0 | 1 | Z | X |
| 7 | 0 | 1 | X | X |
| 8 | 0 | Z | 0 | 0 |
| 9 | 0 | Z | 1 | X |
| 10 | 0 | Z | Z | X |
| 11 | 0 | Z | X | X |
| 12 | 0 | X | 0 | 0 |
| 13 | 0 | X | 1 | X |
| 14 | 0 | X | Z | X |
| 15 | 0 | X | X | X |
| 16 | 1 | 0 | 0 | 1 |
| 17 | 1 | 0 | 1 | 0 |
| 18 | 1 | 0 | Z | X |
| 19 | 1 | 0 | X | X |
| 20 | 1 | 1 | 0 | 1 |
| 21 | 1 | 1 | 1 | 1 |
| 22 | 1 | 1 | Z | X |
| 23 | 1 | 1 | X | X |
| 24 | 1 | Z | 0 | 1 |
| 25 | 1 | Z | 1 | X |
| 26 | 1 | Z | Z | X |
| 27 | 1 | Z | X | X |
| 28 | 1 | X | 0 | 1 |
| 29 | 1 | X | 1 | X |
| 30 | 1 | X | Z | X |
| 31 | 1 | X | X | X |
| 32 | Z | 0 | 0 | X |
| 33 | Z | 0 | 1 | 0 |
| 34 | Z | 0 | Z | X |
| 35 | Z | 0 | X | X |
| 36 | Z | 1 | 0 | X |
| 37 | Z | 1 | 1 | 1 |
| 38 | Z | 1 | Z | X |
| 39 | Z | 1 | X | X |
| 40 | Z | Z | 0 | X |
| 41 | Z | Z | 1 | X |
| 42 | Z | Z | Z | X |
| 43 | Z | Z | X | X |
| 44 | Z | X | 0 | X |
| 45 | Z | X | 1 | X |
| 46 | Z | X | Z | X |
| 47 | Z | X | X | X |
| 48 | X | 0 | 0 | X |
| 49 | X | 0 | 1 | 0 |
| 50 | X | 0 | Z | X |
| 51 | X | 0 | X | X |
| 52 | X | 1 | 0 | X |
| 53 | X | 1 | 1 | 1 |
| 54 | X | 1 | Z | X |
| 55 | X | 1 | X | X |
| 56 | X | Z | 0 | X |
| 57 | X | Z | 1 | X |
| 58 | X | Z | Z | X |
| 59 | X | Z | X | X |
| 60 | X | X | 0 | X |
| 61 | X | X | 1 | X |
| 62 | X | X | Z | X |
| 63 | X | X | X | X |