Appendix

Contents

1 Numbers **6**

1.1 Operator 6

1.2 ConvertFloatToInt 7

2 Integers **7**

2.1 int 7

2.2 common 7

3 Float **7**

3.1 FloatAdd 7

3.1.1 TestFloatAdd 8

3.2 FloatSqrt 8

3.2.1 TestFloatSqrt 8

3.3 FloatDiv 9

3.3.1 TestFloatDiv 9

3.4 FloatMul 9

3.4.1 TestFloatMul 9

3.5 FloatSub 9

3.5.1 TestFloatSub 10

3.6 FloatNeg 10

3.6.1 TestFloatNeg 10

3.7 float 11

3.8 common 11

3.9 FloatSqrt.c 12

4 Real **13**

4.1 RealAdd . . . 13

4.1.1 TestRealAdd . . . 13

4.2 RealSqrt . . . 14

4.2.1 TestRealSqrt . . . 14

4.3 RealDiv . . . 14

4.3.1 TestRealDiv . . . 15

4.4 RealMul . . . 15

4.4.1 TestRealMul . . . 15

4.5 RealSub . . . 15

4.5.1 TestRealSub . . . 16

4.6 RealNeg . . . 16

4.6.1 TestRealNeg . . . 16

4.7 RealOperator . . . 17

4.7.1 TestRealNeg . . . 17

4.8 real . . . 17

4.9 common . . . 17

5 Vector **18**

5.1 VectorAdd . . . 18

5.1.1 TestVectorAdd . . . 18

5.2 VectorLength . . . 18

5.2.1 TestVectorLength . . . 19

5.3 VectorNormalize . . . 19

5.3.1 TestVectorNormalize . . . 20

5.4 VectorDivReal . . . 20

5.4.1 TestVectorDivReal . . . 20
5.5 VectorMulReal . . . 21
5.5.1 TestVectorMulReal . . . 21
5.6 VectorSub . . . 21
5.6.1 TestVectorSub . . . 22
5.7 VectorDot . . . 22
5.7.1 TestVectorDot . . . 23
5.8 VectorNeg . . . 23
5.8.1 TestVectorNeg . . . 23
5.9 Common . . . 23

6 Geometry **24**
6.1 Rectangle . . . 24
6.2 Circle . . . 25
6.2.1 TestCircle . . . 28
6.3 CircleOctant . . . 29
6.3.1 TestCircleOctant . . . 31

7 Display **32**
7.1 DrawerCircle . . . 32
7.1.1 TestDrawerCircle . . . 33
7.2 VGAController . . . 35

8 Memory **36**
8.1 ZBT . . . 36
8.1.1 TestZBT . . . 38
8.2 BufferSwitcher . . . 39
8.2.1 TestBufferSwitcher . . . 41

9 Control **43**

9.1 InteractionScheduler 43

9.1.1 TestInteractionScheduler 45

9.2 InteractionScheduler.py 46

10 Physics **46**

10.1 GravitationalAccelerator 46

10.1.1 TestGravitationalAccelerator 49

10.2 Constants 50

11 User Interface: PS/2 Mouse **50**

11.1 PS2Mouse 50

11.1.1 TestPS2Mouse 53

11.2 PS2MouseController 55

11.2.1 TestPS2MouseController 58

11.3 PS2Sender 58

11.3.1 TestPS2Sender 60

11.4 PS2Receiver 62

11.4.1 TestPS2Receiver 63

11.5 PS2MouseDecoder 63

11.5.1 TestPS2MouseDecoder 65

12 LabKit **65**

12.1 LabKitPS2Mouse 65

12.2 LabKitZBT 72

12.3 LabKitScheduler 79

12.4 LabKitDrawerCircle 87

12.4.1 TestLabKitDrawerCircle 87

12.5 LabKitCircleDrawerDoubleBuffered 92

12.5.1 TestLabKitCircleDrawerDoubleBuffered 92

12.6 LabKitScheduler 92

13 Utilities **101**

13.1 Debouncer 101

13.2 Enabler 102

13.3 EnableCounter 102

13.4 LevelToPulse 103

13.4.1 LevelToPulseNeg 104

13.4.2 LevelToPulsePos 104

13.5 Delay 104

13.5.1 TestDelay 105

13.6 FIFO 106

13.6.1 TestFIFO 107

1 Numbers

1.1 Operator

```
‘ifdef HOME
    ‘include "src/synthesis/util/Delay.v"
‘endif

module ‘OPERATOR_NAME
(
    input clk,

    input  ‘TYPE_ARG1 arg1,

    ‘ifndef UNARY
        input  ‘TYPE_ARG2 arg2,
    ‘endif

    output ‘TYPE_RESULT result
);
    parameter delay = ‘DELAY;

    ‘ifdef HOME

        ‘OPERATION

        ‘ifdef DELAY_MODULE
            Delay #(.delay(delay), .width(‘DELAY_BITS)) delayResult
            (
                clk,
                ‘RESULT_TEMP,
                result
            );
        ‘endif

    ‘else
        ‘ifdef OPERATOR_NAME_SMALL
            ‘OPERATOR_NAME_SMALL operator
            (
                arg1,
                ‘ifndef UNARY arg2, ‘endif
                clk,
                result
            );
        ‘else
            ‘OPERATION
        ‘endif
    ‘endif

    ‘ifndef DELAY
        ‘define DELAY 0
    ‘endif

endmodule
```

1.2 ConvertFloatToInt

2 Integers

2.1 int

```
‘ifndef INCLUDED_D73AAE02_2071_4D9A_9968_1C19C6220786

    ‘define INCLUDED_D73AAE02_2071_4D9A_9968_1C19C6220786

    ‘include "src/synthesis/math/number/int/common.v"

‘endif
```

2.2 common

```
‘ifndef INCLUDED_EDD2DEA7_D917_4799_A0AE_03F07EE39AC4

    ‘define INCLUDED_EDD2DEA7_D917_4799_A0AE_03F07EE39AC4

    //////////////////
    ////Discussion////
    //////////////////


    ////////////////////////////
    ////Interesting numbers////
    ////////////////////////////

    ‘define INT_BITS 32
    ‘define INT_MSB  (‘INT_BITS - 1)


    ////////////////////////////
    ////Declaring variables////
    ////////////////////////////

    ‘define INT [‘INT_MSB:0]

‘endif
```

3 Float

3.1 FloatAdd

```
‘ifndef INCLUDED_FBE2E266_2C65_4AB7_90A5_9C4FF281C2A6
    ‘define INCLUDED_FBE2E266_2C65_4AB7_90A5_9C4FF281C2A6

    ‘define OPERATOR             +
    ‘define OPERATOR_NAME       FloatAdd
    ‘define OPERATOR_NAME_SMALL float_add

    ‘define FloatAdd_DELAY  13
```

```
    `define DELAY          `FloatAdd_DELAY
    `define DELAY_MODULE

    `undef UNARY

    `include "src/synthesis/math/number/real/float/FloatOperator.v"

`endif
//newline required
```

3.1.1 TestFloatAdd

```
`include "src/synthesis/math/number/real/float/FloatAdd.v"
`include "src/test/math/number/real/float/common.v"
`include "src/test/math/number/real/Operator.v"
// needs a newline
```

3.2 FloatSqrt

```
`ifndef INCLUDED_77159C4C_99C1_406A_831D_E2522C5F2E27
    `define INCLUDED_77159C4C_99C1_406A_831D_E2522C5F2E27

    `define OPERATOR_NAME       FloatSqrt
    `define OPERATOR_NAME_SMALL float_sqrt

    `define FLOAT_OPERATION                                                 \
        real resultRealTemp;                                                \
                                                                            \
            always @ (arg1)                                                 \
                begin                                                       \
                    $FloatSqrt($bitstoreal(arg1), resultRealTemp);          \
                    resultReal = resultRealTemp; /*iverilog really sucks*/  \
                end

    `ifdef HOME
        `define FloatSqrt_DELAY 32
    `else
        `define FloatSqrt_DELAY (`FLOAT_FRACTION_BITS+4)
    `endif

    `define DELAY `FloatSqrt_DELAY
    `define DELAY_MODULE

    `define UNARY

    `include "src/synthesis/math/number/real/float/FloatOperator.v"

`endif
//newline required
```

3.2.1 TestFloatSqrt

```
`include "src/synthesis/math/number/real/float/FloatSqrt.v"
`include "src/test/math/number/real/float/common.v"

`define UNARY

`include "src/test/math/number/real/Operator.v"
// needs a newline
```

3.3 FloatDiv

```
‘ifndef INCLUDED_07623DD3_7A9A_4734_B9E4_C3952846744B
    ‘define INCLUDED_07623DD3_7A9A_4734_B9E4_C3952846744B

    ‘define OPERATOR             /
    ‘define OPERATOR_NAME       FloatDiv
    ‘define OPERATOR_NAME_SMALL float_div

    ‘include "src/synthesis/math/number/real/float/common.v"

    ‘ifdef HOME
        ‘define FloatDiv_DELAY 32
    ‘else
        ‘define FloatDiv_DELAY (‘FLOAT_FRACTION_BITS+4)
    ‘endif

    ‘define DELAY ‘FloatDiv_DELAY
    ‘define DELAY_MODULE

    ‘undef UNARY

    ‘include "src/synthesis/math/number/real/float/FloatOperator.v"

‘endif
//newline required
```

3.3.1 TestFloatDiv

```
‘include "src/synthesis/math/number/real/float/FloatDiv.v"
‘include "src/test/math/number/real/float/common.v"
‘include "src/test/math/number/real/Operator.v"
// needs a newline
```

3.4 FloatMul

```
‘ifndef INCLUDED_31E471B2_4587_4881_B9ED_C95CFF464148
    ‘define INCLUDED_31E471B2_4587_4881_B9ED_C95CFF464148

    ‘define OPERATOR             *
    ‘define OPERATOR_NAME       FloatMul
    ‘define OPERATOR_NAME_SMALL float_mul

    ‘define FloatMul_DELAY  8
    ‘define DELAY          ‘FloatMul_DELAY
    ‘define DELAY_MODULE

    ‘undef UNARY

    ‘include "src/synthesis/math/number/real/float/FloatOperator.v"

‘endif
//newline required
```

3.4.1 TestFloatMul

3.5 FloatSub

```
‘ifndef INCLUDED_FC303188_0BA8_4D98_9724_658BBE7481FE
```

```
    `define INCLUDED_FC303188_0BA8_4D98_9724_658BBE7481FE

    `define OPERATOR             -
    `define OPERATOR_NAME       FloatSub
    `define OPERATOR_NAME_SMALL float_sub

    `define FloatSub_DELAY  13
    `define DELAY          `FloatSub_DELAY
    `define DELAY_MODULE

    `undef UNARY

    `include "src/synthesis/math/number/real/float/FloatOperator.v"

`endif
//newline required
```

3.5.1 TestFloatSub

```
`include "src/synthesis/math/number/real/float/FloatSub.v"
`include "src/test/math/number/real/float/common.v"
`include "src/test/math/number/real/Operator.v"
// needs a newline
```

3.6 FloatNeg

```
`ifndef INCLUDED_6DAB2FA6_2629_4B93_821F_D38C3A19C7AD
    `define INCLUDED_6DAB2FA6_2629_4B93_821F_D38C3A19C7AD

    `define OPERATOR_NAME FloatNeg
    `undef  OPERATOR_NAME_SMALL

    `define FLOAT_OPERATION assign result = {~arg1[`FLOAT_SIGN], arg1[`FLOAT_SIGN-1:0]};

    //// No delay ////

    `define FloatNeg_DELAY  0
    `define DELAY          `FloatNeg_DELAY
    `undef  DELAY_MODULE

    `define UNARY

    `include "src/synthesis/math/number/real/float/FloatOperator.v"

`endif
// newline required
```

3.6.1 TestFloatNeg

```
`include "src/synthesis/math/number/real/float/FloatNeg.v"
`include "src/test/math/number/real/float/common.v"

`define UNARY

`include "src/test/math/number/real/Operator.v"
// needs a newline
```

3.7 float

```
‘ifndef INCLUDED_6CE680D5_E33A_459E_9900_97617F468F65

    ‘define INCLUDED_6CE680D5_E33A_459E_9900_97617F468F65

    ‘include "src/synthesis/math/number/real/float/FloatAdd.v"
    ‘include "src/synthesis/math/number/real/float/FloatSub.v"
    ‘include "src/synthesis/math/number/real/float/FloatMul.v"
    ‘include "src/synthesis/math/number/real/float/FloatDiv.v"
    ‘include "src/synthesis/math/number/real/float/FloatNeg.v"
    ‘include "src/synthesis/math/number/real/float/FloatSqrt.v"

‘endif
//newline required
```

3.8 common

```
‘ifndef INCLUDED_F31BEDE8_45C3_4FFF_AD63_C66E35743BD1

    ‘define INCLUDED_F31BEDE8_45C3_4FFF_AD63_C66E35743BD1

    ////////////////////
    //// Discussion ////
    ////////////////////


    ////////////////////////////
    //// Interesting numbers ////
    ////////////////////////////

    ‘ifdef HOME
        ‘define FLOAT_BITS          64
        ‘define FLOAT_FRACTION_BITS 52
    ‘else
        ‘define FLOAT_BITS          32
        ‘define FLOAT_FRACTION_BITS 23
    ‘endif

    ‘define FLOAT_EXPONENT_BITS (‘FLOAT_BITS - ‘FLOAT_FRACTION_BITS)

    ‘define FLOAT_MSB  (‘FLOAT_BITS - 1)


    ////////////////////////////
    //// Declaring variables ////
    ////////////////////////////

    ‘define FLOAT [‘FLOAT_MSB:0]


    ////////////////////////
    //// Interpretation ////
    ////////////////////////

    ‘define FLOAT_SIGN ‘FLOAT_MSB

    ‘define FLOAT_EXPONENT_MSB (‘FLOAT_SIGN - 1)
    ‘define FLOAT_EXPONENT_LSB (‘FLOAT_EXPONENT_MSB - ‘FLOAT_EXPONENT_BITS + 1)

    ‘define FLOAT_FRACTION_MSB (‘FLOAT_EXPONENT_LSB - 1)
    ‘define FLOAT_FRACTION_LSB (‘FLOAT_FRACTION_MSB - ‘FLOAT_FRACTION_BITS + 1)

    ‘define FLOAT_EXPONENT  ‘FLOAT_EXPONENT_MSB:‘FLOAT_EXPONENT_LSB
    ‘define FLOAT_FRACTION  ‘FLOAT_FRACTION_MSB:‘FLOAT_FRACTION_LSB
```

```
`endif
//newline necessary
```

3.9 FloatSqrt.c

```
# include <vpi_user.h>
# include <math.h>

static PLI_INT32 compiletf_FloatSqrt(PLI_BYTE8* data)
{
    vpiHandle   callHandle,
                argHandles,
                argHandle  = 0;

    PLI_UINT32  type;

    /* get argument handles */
    callHandle = vpi_handle(vpiSysTfCall, 0);
    argHandles = vpi_iterate(vpiArgument, callHandle);

    if (
            argHandles
            && (argHandle = vpi_scan(argHandles)) && ((type = vpi_get(vpiType, argHandle)) == vpiRealVar || type == vpiRealVal)
            && (argHandle = vpi_scan(argHandles)) && vpi_get(vpiType, argHandle) == vpiRealVar
            && !vpi_scan(argHandles)
       )
        return 0;
    else
    {
   vpi_printf("ERROR: %s requires 2 reals: (const/var, var).\n", vpi_get_str(vpiName, callHandle));

   if (argHandle)
           vpi_free_object(argHandles);

   return -1;
    }
}

static PLI_INT32 calltf_FloatSqrt(PLI_BYTE8* user)
{
    vpiHandle   callHandle,
                argHandles,
                argHandle;

    s_vpi_value argValue;

    /* get input handle */
    callHandle = vpi_handle(vpiSysTfCall, 0);
    argHandles = vpi_iterate(vpiArgument, callHandle);
    argHandle  = vpi_scan(argHandles);

    /* get input value */
    argValue.format = vpiRealVal;
    vpi_get_value(argHandle, &argValue);

    /* compute square root */
    argValue.value.real = sqrt(argValue.value.real);

    /* get ouput handle */
    argHandle = vpi_scan(argHandles);

    /* set output value */
    vpi_put_value(argHandle, &argValue, 0, vpiNoDelay);

    /* free iterator */
```

```
    vpi_free_object(argHandles);

    return 0;
}

static int sizetf_64(PLI_BYTE8* unused)
{
    return 64;
}

static void register_FloatSqrt()
{
    s_vpi_systf_data tf_data;

    tf_data.type      = vpiSysTask;
    tf_data.user_data = 0;
    tf_data.tfname    = "$FloatSqrt";
    tf_data.sizetf    = sizetf_64;
    tf_data.compiletf = compiletf_FloatSqrt;
    tf_data.calltf    = calltf_FloatSqrt;
    vpi_register_systf(&tf_data);
}

void (*vlog_startup_routines[])() =
{
    register_FloatSqrt,
    0
};
```

4 Real

4.1 RealAdd

```
`ifndef INCLUDED_334EA0EC_969E_4740_ADD0_7F4023DA4E80
    `define INCLUDED_334EA0EC_969E_4740_ADD0_7F4023DA4E80

    `ifdef REAL_FIXED
        `include "src/synthesis/math/number/real/fixed/FixedAdd.v"

        `define OPERATOR       FixedAdd
        `define RealAdd_DELAY `FixedAdd_DELAY
    `else
        `include "src/synthesis/math/number/real/float/FloatAdd.v"

        `define OPERATOR       FloatAdd
        `define RealAdd_DELAY `FloatAdd_DELAY
    `endif

    `define OPERATOR_NAME  RealAdd
    `undef  DELAY_MODULE
    `undef  UNARY

    `include "src/synthesis/math/number/real/RealOperator.v"

`endif
//newline required
```

4.1.1 TestRealAdd

```
`include "src/synthesis/math/number/real/RealAdd.v"
```

```
`include "src/test/math/number/real/common.v"
`include "src/test/math/number/real/Operator.v"
// needs a newline
```

4.2 RealSqrt

```
`ifndef INCLUDED_C87A3290_1EE5_4D19_8E6A_81902B4D6774
    `define INCLUDED_C87A3290_1EE5_4D19_8E6A_81902B4D6774

    `ifdef REAL_FIXED
        `include "src/synthesis/math/number/real/fixed/FixedSqrt.v"

        `define OPERATOR        FixedSqrt
        `define RealSqrt_DELAY `FixedSqrt_DELAY
    `else
        `include "src/synthesis/math/number/real/float/FloatSqrt.v"

        `define OPERATOR        FloatSqrt
        `define RealSqrt_DELAY `FloatSqrt_DELAY
    `endif

    `define OPERATOR_NAME  RealSqrt
    `undef  DELAY_MODULE
    `define UNARY

    `include "src/synthesis/math/number/real/RealOperator.v"

`endif
//newline required
```

4.2.1 TestRealSqrt

```
`include "src/synthesis/math/number/real/RealSqrt.v"
`include "src/test/math/number/real/common.v"

`define UNARY

`include "src/test/math/number/real/Operator.v"
// needs a newline
```

4.3 RealDiv

```
`ifndef INCLUDED_F2A38844_5454_4B86_B2B6_27DE8C639B20
    `define INCLUDED_F2A38844_5454_4B86_B2B6_27DE8C639B20

    `ifdef REAL_FIXED
        `include "src/synthesis/math/number/real/fixed/FixedDiv.v"

        `define OPERATOR       FixedDiv
        `define RealDiv_DELAY `FixedDiv_DELAY
    `else
        `include "src/synthesis/math/number/real/float/FloatDiv.v"

        `define OPERATOR       FloatDiv
        `define RealDiv_DELAY `FloatDiv_DELAY
    `endif

    `define OPERATOR_NAME  RealDiv
```

```
    `undef  DELAY_MODULE
    `undef  UNARY

    `include "src/synthesis/math/number/real/RealOperator.v"

`endif
//newline required
```

4.3.1 TestRealDiv

```
`include "src/synthesis/math/number/real/RealDiv.v"
`include "src/test/math/number/real/common.v"
`include "src/test/math/number/real/Operator.v"
// needs a newline
```

4.4 RealMul

```
`ifndef INCLUDED_96776E47_58FC_4879_84D8_7D942B3EE11C
    `define INCLUDED_96776E47_58FC_4879_84D8_7D942B3EE11C

    `ifdef REAL_FIXED
        `include "src/synthesis/math/number/real/fixed/FixedMul.v"

        `define OPERATOR       FixedMul
        `define RealMul_DELAY `FixedMul_DELAY
    `else
        `include "src/synthesis/math/number/real/float/FloatMul.v"

        `define OPERATOR       FloatMul
        `define RealMul_DELAY `FloatMul_DELAY
    `endif

    `define OPERATOR_NAME  RealMul
    `undef  DELAY_MODULE
    `undef  UNARY

    `include "src/synthesis/math/number/real/RealOperator.v"

`endif
//newline required
```

4.4.1 TestRealMul

4.5 RealSub

```
`ifndef INCLUDED_DDDA5611_AB8A_452E_AE13_AF3023E933AC
    `define INCLUDED_DDDA5611_AB8A_452E_AE13_AF3023E933AC

    `ifdef REAL_FIXED
        `include "src/synthesis/math/number/real/fixed/FixedSub.v"

        `define OPERATOR       FixedSub
        `define RealSub_DELAY `FixedSub_DELAY
    `else
        `include "src/synthesis/math/number/real/float/FloatSub.v"

        `define OPERATOR       FloatSub
```

```
        `define RealSub_DELAY `FloatSub_DELAY
    `endif

    `define OPERATOR_NAME  RealSub
    `undef  DELAY_MODULE
    `undef  UNARY

    `include "src/synthesis/math/number/real/RealOperator.v"

`endif
//newline required
```

4.5.1 TestRealSub

```
`include "src/synthesis/math/number/real/RealSub.v"
`include "src/test/math/number/real/common.v"
`include "src/test/math/number/real/Operator.v"
// needs a newline
```

4.6 RealNeg

```
`ifndef INCLUDED_FA7301D7_2AC7_462A_95C1_E9658EB7AF26
    `define INCLUDED_FA7301D7_2AC7_462A_95C1_E9658EB7AF26

    `ifdef REAL_FIXED
        `include "src/synthesis/math/number/real/fixed/FixedNeg.v"

        `define OPERATOR       FixedNeg
        `define RealNeg_DELAY `FixedNeg_DELAY
    `else
        `include "src/synthesis/math/number/real/float/FloatNeg.v"

        `define OPERATOR       FloatNeg
        `define RealNeg_DELAY `FloatNeg_DELAY
    `endif

    `define OPERATOR_NAME  RealNeg
    `undef  DELAY_MODULE
    `define UNARY

    `include "src/synthesis/math/number/real/RealOperator.v"

`endif
//newline required
```

4.6.1 TestRealNeg

```
`include "src/synthesis/math/number/real/RealNeg.v"
`include "src/test/math/number/real/common.v"

`define UNARY

`include "src/test/math/number/real/Operator.v"
// needs a newline
```

4.7 RealOperator

```
‘include "src/synthesis/math/number/real/common.v"

‘define TYPE_ARG1   ‘REAL
‘define TYPE_ARG2   ‘REAL
‘define TYPE_RESULT ‘REAL

‘undef OPERATOR_NAME_SMALL

‘ifdef UNARY
    ‘define OPERATION ‘OPERATOR operator(clk, arg1, result);
‘else
    ‘define OPERATION ‘OPERATOR operator(clk, arg1, arg2, result);
‘endif

‘define DELAY_NONE
‘define DELAY_BITS ‘REAL_BITS

‘include "src/synthesis/math/number/Operator.v"
//newline required
```

4.7.1 TestRealNeg

4.8 real

```
‘ifndef INCLUDED_777D558F_E679_49CB_B058_982907A91125

    ‘define INCLUDED_777D558F_E679_49CB_B058_982907A91125

    ‘include "src/synthesis/math/number/real/RealAdd.v"
    ‘include "src/synthesis/math/number/real/RealSub.v"
    ‘include "src/synthesis/math/number/real/RealMul.v"
    ‘include "src/synthesis/math/number/real/RealDiv.v"
    ‘include "src/synthesis/math/number/real/RealNeg.v"
    ‘include "src/synthesis/math/number/real/RealSqrt.v"

‘endif
//newline required
```

4.9 common

```
‘ifndef INCLUDED_4F7D6E3E_EB1E_47CA_A624_4BEC78980628

    ‘define INCLUDED_4F7D6E3E_EB1E_47CA_A624_4BEC78980628

    ‘ifdef REAL_FIXED

        ‘include "src/synthesis/math/number/real/fixed/common.v"

        ‘define REAL_BITS ‘FIXED_BITS
        ‘define REAL      ‘FIXED

    ‘else

        ‘include "src/synthesis/math/number/real/float/common.v"

        ‘define REAL_BITS ‘FLOAT_BITS
        ‘define REAL      ‘FLOAT
```

```
    `endif

`endif
```

5 Vector

5.1 VectorAdd

```
`ifndef INCLUDED_4FD4F259_AC28_4017_AE0E_36BE568637FA
    `define INCLUDED_4FD4F259_AC28_4017_AE0E_36BE568637FA

    `include "src/synthesis/math/vector/common.v"
    `include "src/synthesis/math/number/real/RealAdd.v"

    `define VectorAdd_DELAY `RealAdd_DELAY

    module VectorAdd
    (
        input          clk,

        input  `VECTOR v1,
        input  `VECTOR v2,

        output `VECTOR result
    );

        parameter delay = `VectorAdd_DELAY;

        wire `REAL x, y;

        RealAdd xAdd(clk, v1[`X], v2[`X], x);
        RealAdd yAdd(clk, v1[`Y], v2[`Y], y);

        assign result = {x, y};

    endmodule

`endif
```

5.1.1 TestVectorAdd

```
`include "src/synthesis/math/vector/VectorAdd.v"

`define OPERATOR_NAME Add

`include "src/test/math/vector/Operator.v"
// needs a newline
```

5.2 VectorLength

```
`ifndef INCLUDED_98207ECB_01ED_4A6C_9442_745BADE21175
    `define INCLUDED_98207ECB_01ED_4A6C_9442_745BADE21175

    `include "src/synthesis/math/vector/common.v"
```

```
    `include "src/synthesis/math/vector/VectorDot.v"
    `include "src/synthesis/math/number/real/RealSqrt.v"

    `define VectorLength_DELAY (`VectorDot_DELAY + `RealSqrt_DELAY)

    module VectorLength
    (
        input          clk,

        input  `VECTOR v1,

        output `REAL   result
    );

        parameter delay = `VectorLength_DELAY;

        wire `REAL lengthSquared;

        VectorDot getLengthSquared(clk, v1, v1, lengthSquared);

        RealSqrt sqrt(clk, lengthSquared, result);

    endmodule

`endif
```

5.2.1 TestVectorLength

```
`include "src/synthesis/math/vector/VectorLength.v"

`define OPERATOR_NAME Length
`define OUTPUT_REAL
`define UNARY

`include "src/test/math/vector/Operator.v"
// needs a newline
```

5.3 VectorNormalize

```
`ifndef INCLUDED_55B10023_A830_4273_B6D8_3AD59CFB625F
    `define INCLUDED_55B10023_A830_4273_B6D8_3AD59CFB625F

    `include "src/synthesis/math/vector/common.v"
    `include "src/synthesis/math/vector/VectorLength.v"
    `include "src/synthesis/math/vector/VectorDivReal.v"

    `define VectorNormalize_DELAY (`VectorLength_DELAY + `VectorDivReal_DELAY)

    module VectorNormalize
    (
        input          clk,

        input  `VECTOR v1,

        output `VECTOR result
    );

        parameter delay = `VectorNormalize_DELAY;

        wire `REAL length;

        VectorLength  getLength(clk, v1, length);
```

```
        VectorDivReal divideByLength(clk, v1, length, result);

    endmodule

‘endif
```

5.3.1 TestVectorNormalize

```
‘include "src/synthesis/math/vector/VectorNormalize.v"

‘define OPERATOR_NAME Normalize
‘define UNARY

‘include "src/test/math/vector/Operator.v"
// needs a newline
```

5.4 VectorDivReal

```
‘ifndef INCLUDED_F7C037C6_40C2_4C78_8339_13B09608B641
    ‘define INCLUDED_F7C037C6_40C2_4C78_8339_13B09608B641

    ‘include "src/synthesis/math/vector/common.v"
    ‘include "src/synthesis/math/number/real/RealDiv.v"

    ‘define VectorDivReal_DELAY ‘RealDiv_DELAY

    module VectorDivReal
    (
        input          clk,

        input  ‘VECTOR v1,
        input  ‘REAL    s,

        output ‘VECTOR result
    );

        parameter delay = ‘VectorDivReal_DELAY;

        wire ‘REAL x, y;

        RealDiv xDiv(clk, v1[‘X], s, x);
        RealDiv yDiv(clk, v1[‘Y], s, y);

        assign result = {x, y};

    endmodule

‘endif
```

5.4.1 TestVectorDivReal

```
‘include "src/synthesis/math/vector/VectorDivReal.v"

‘define OPERATOR_NAME DivReal
‘define INPUT_REAL

‘include "src/test/math/vector/Operator.v"
// needs a newline
```

5.5 VectorMulReal

```
‘ifndef INCLUDED_113F08A9_0826_4CED_814D_FB6169090254
    ‘define INCLUDED_113F08A9_0826_4CED_814D_FB6169090254

    ‘include "src/synthesis/math/vector/common.v"
    ‘include "src/synthesis/math/number/real/RealMul.v"

    ‘define VectorMulReal_DELAY ‘RealMul_DELAY

    module VectorMulReal
    (
        input          clk,

        input  ‘VECTOR v1,
        input  ‘REAL    s,

        output ‘VECTOR result
    );

        parameter delay = ‘VectorMulReal_DELAY;

        wire ‘REAL x, y;

        RealMul xMul(clk, v1[‘X], s, x);
        RealMul yMul(clk, v1[‘Y], s, y);

        assign result = {x, y};

    endmodule

‘endif
```

5.5.1 TestVectorMulReal

```
‘include "src/synthesis/math/vector/VectorMulReal.v"

‘define OPERATOR_NAME MulReal
‘define INPUT_REAL

‘include "src/test/math/vector/Operator.v"
// needs a newline
```

5.6 VectorSub

```
‘ifndef INCLUDED_4A22ED55_CEDD_474C_A4F4_4BE7446625CB
    ‘define INCLUDED_4A22ED55_CEDD_474C_A4F4_4BE7446625CB

    ‘include "src/synthesis/math/vector/common.v"
    ‘include "src/synthesis/math/number/real/RealSub.v"

    ‘define VectorSub_DELAY ‘RealSub_DELAY

    module VectorSub
    (
        input          clk,

        input  ‘VECTOR v1,
        input  ‘VECTOR v2,

        output ‘VECTOR result
```

```
    );

        parameter delay = `VectorSub_DELAY;

        wire `REAL x, y;

        RealSub xSub(clk, v1[`X], v2[`X], x);
        RealSub ySub(clk, v1[`Y], v2[`Y], y);

        assign result = {x, y};

    endmodule

`endif
```

5.6.1 TestVectorSub

```
`include "src/synthesis/math/vector/VectorSub.v"

`define OPERATOR_NAME Sub

`include "src/test/math/vector/Operator.v"
// needs a newline
```

5.7 VectorDot

```
`ifndef INCLUDED_2883388D_AC55_4AF3_83A1_4E5489979177
    `define INCLUDED_2883388D_AC55_4AF3_83A1_4E5489979177

    `include "src/synthesis/math/vector/common.v"
    `include "src/synthesis/math/number/real/RealMul.v"
    `include "src/synthesis/math/number/real/RealAdd.v"

    `define VectorDot_DELAY (`RealMul_DELAY + `RealAdd_DELAY)

    module VectorDot
    (
        input          clk,

        input  `VECTOR v1,
        input  `VECTOR v2,

        output `REAL   result
    );

        parameter delay = `VectorDot_DELAY;

        wire `REAL xMultiplied,
                   yMultiplied;

        // We could use untruncated temporaries here
        // in order to improve upon numerical error.

        RealMul xMul(clk, v1[`X], v2[`X], xMultiplied);
        RealMul yMul(clk, v1[`Y], v2[`Y], yMultiplied);

        RealAdd  mulAdd(clk, xMultiplied, yMultiplied, result);

    endmodule

`endif
```

5.7.1 TestVectorDot

```
`include "src/synthesis/math/vector/VectorDot.v"

`define OPERATOR_NAME Dot
`define OUTPUT_REAL

`include "src/test/math/vector/Operator.v"
// needs a newline
```

5.8 VectorNeg

```
`ifndef INCLUDED_BC238F0F_328B_40FB_8903_67CBCD60C072
    `define INCLUDED_BC238F0F_328B_40FB_8903_67CBCD60C072

    `include "src/synthesis/math/vector/common.v"
    `include "src/synthesis/math/number/real/RealNeg.v"

    `define VectorNeg_DELAY `RealNeg_DELAY

    module VectorNeg
    (
        input          clk,

        input  `VECTOR v1,

        output `VECTOR result
    );

        parameter delay = `VectorNeg_DELAY;

        // We could use untruncated temporaries here
        // in order to improve upon numerical error.

        wire `REAL x, y;

        RealNeg yNeg(clk, v1[`X], x);
        RealNeg xNeg(clk, v1[`Y], y);

        assign result = {x, y};

    endmodule

`endif
```

5.8.1 TestVectorNeg

```
`include "src/synthesis/math/vector/VectorNeg.v"

`define OPERATOR_NAME Neg
`define UNARY

`include "src/test/math/vector/Operator.v"
// needs a newline
```

5.9 Common

```
`ifndef INCLUDED_D2FFAD62_902A_43C3_B0AB_3F5347D82574
```

```
    `define INCLUDED_D2FFAD62_902A_43C3_B0AB_3F5347D82574

    `include "src/synthesis/math/number/real/common.v"

    ////////////////////
    //// Discussion ////
    ////////////////////
    //
    // We could do calculations in 3 dimensions simply by defining
    // vectors in this file to have 3 dimensions.
    //

    ////////////////////////////
    //// Interesting numbers ////
    ////////////////////////////

    `define VECTOR_BITS (`REAL_BITS + `REAL_BITS)
    `define VECTOR_MSB  (`VECTOR_BITS - 1)

    `define VECTOR_X_MSB `VECTOR_MSB
    `define VECTOR_X_LSB (`VECTOR_X_MSB - `REAL_BITS + 1)

    `define VECTOR_Y_MSB (`VECTOR_X_LSB - 1)
    `define VECTOR_Y_LSB (`VECTOR_Y_MSB - `REAL_BITS + 1)

    ////////////////////////////
    //// Declaring variables ////
    ////////////////////////////

    `define VECTOR [`VECTOR_MSB:0]


    ////////////////////////
    //// Interpretation ////
    ////////////////////////

    `define X `VECTOR_X_MSB:`VECTOR_X_LSB
    `define Y `VECTOR_Y_MSB:`VECTOR_Y_LSB

`endif
```

6 Geometry

6.1 Rectangle

```
////////////////////////////////////////////////////////////////////////////////
//
// Rectangle: generate rectangle on screen
//
// Checks wether a given coordinate is within a rectangle.
//
////////////////////////////////////////////////////////////////////////////////

`ifndef INCLUDE_4AADD270_E87B_48F4_B443_1A0751AEC7FB
    `define INCLUDE_4AADD270_E87B_48F4_B443_1A0751AEC7FB

    module Rectangle
    #(parameter

        bitsX  = 10,
        bitsY  = 10
    )
```

```
    (
        input               reset,    //HIGH: don't display anything

        input  [bitsX-1:0] originX,  //The x-coordinate of the top left of the rectangle
        input  [bitsY-1:0] originY,  //The y-coordinate of the top left of the rectangle

        input  [bitsX-1:0] width,
        input  [bitsY-1:0] height,

        input  [bitsX-1:0] x,   //The x-coordinate to test.
        input  [bitsY-1:0] y,   //The y-coordinate to test.

        output              isInside    //location (x, y) is inside the rectangle.
    );

        wire xInside = (x >= originX) && (x < (originX+width ));
        wire yInside = (y >= originY) && (y < (originY+height));

        assign isInside = !reset && xInside && yInside;

    endmodule

`endif
```

6.2 Circle

```
`ifndef INCLUDED_5996B2AE_7B30_4A64_A745_65BAB6BB060C
    `define INCLUDED_5996B2AE_7B30_4A64_A745_65BAB6BB060C

    `include "src/synthesis/math/geometry/CircleOctant.v"

    module Circle
    (
        input                   clk,
        input                   reset,

        input      signed `INT centerX,
        input      signed `INT centerY,

        input      signed `INT radius,

        input                   next,

        output reg signed `INT x,
        output reg signed `INT y,

        output reg              done
    );

        wire signed `INT xOctant,
                         yOctant;

        reg getNextOctant;

        CircleOctant drawerCircleOctant
        (
            clk,
            reset,

            radius,

            getNextOctant,

            xOctant,
            yOctant
        );
```

```
reg [2:0] count;  // This regulates the pipeline; one point is calculated per cycle.

always @ (posedge clk)
    begin
        // pulse these signal
        getNextOctant <= 0;
        done          <= 0;

        if (reset)
            begin
                // output the first point: (0, radius); the octant drawer will put this out in the next cycle too.
                count      <= 0;
                x          <= centerX;
                y          <= centerY+radius;
            end
        else if (!done && next)
            begin
                count <= count+1;

                if (xOctant==0)
                    case (count)
                        3'd0:
                            begin
                                x <= centerX;
                                y <= centerY - yOctant;
                            end
                        3'd1:
                            begin
                                x <= centerX + yOctant;
                                y <= centerY;

                                getNextOctant <= 1;
                            end
                        3'd2:
                            begin
                                x <= centerX - yOctant;
                                y <= centerY;

                                count      <= 0;
                            end
                        default:
                            begin
                                x     <= 'INT_BITS'hx;
                                y     <= 'INT_BITS'hx;
                                count <=         3'hx;
                            end
                    endcase
                else if (xOctant==yOctant)
                    case (count)
                        3'd0:
                            begin
                                x <= centerX + xOctant;
                                y <= centerY + yOctant;
                            end
                        3'd1:
                            begin
                                x <= centerX - xOctant;
                                y <= centerY + yOctant;
                            end
                        3'd2:
                            begin
                                x <= centerX + xOctant;
                                y <= centerY - yOctant;

                                getNextOctant <= 1;
                            end
                        3'd3:
                            begin
```

```
                                    x <= centerX - xOctant;
                                    y <= centerY - yOctant;

                                    count      <= 0;
                                end
                            default:
                                begin
                                    x     <= `INT_BITS'hx;
                                    y     <= `INT_BITS'hx;
                                    count <=         3'hx;
                                end
                        endcase
                    else if (xOctant<yOctant)
                        case (count)
                            3'd0:
                                begin
                                    x <= centerX + xOctant;
                                    y <= centerY + yOctant;
                                end
                            3'd1:
                                begin
                                    x <= centerX - xOctant;
                                    y <= centerY + yOctant;
                                end
                            3'd2:
                                begin
                                    x <= centerX + xOctant;
                                    y <= centerY - yOctant;
                                end
                            3'd3:
                                begin
                                    x <= centerX - xOctant;
                                    y <= centerY - yOctant;
                                end
                            3'd4:
                                begin
                                    x <= centerX + yOctant;
                                    y <= centerY + xOctant;
                                end
                            3'd5:
                                begin
                                    x <= centerX - yOctant;
                                    y <= centerY + xOctant;
                                end
                            3'd6:
                                begin
                                    x <= centerX + yOctant;
                                    y <= centerY - xOctant;

                                    getNextOctant <= 1;
                                end
                            3'd7:
                                begin
                                    x <= centerX - yOctant;
                                    y <= centerY - xOctant;

                                    //count      <= 0;          // implicit
                                end
                        endcase
                    else // if (xOctant > yOctant)
                        done <= 1;
                end
            end
        endmodule

    `endif
    // newline required
```

6.2.1 TestCircle

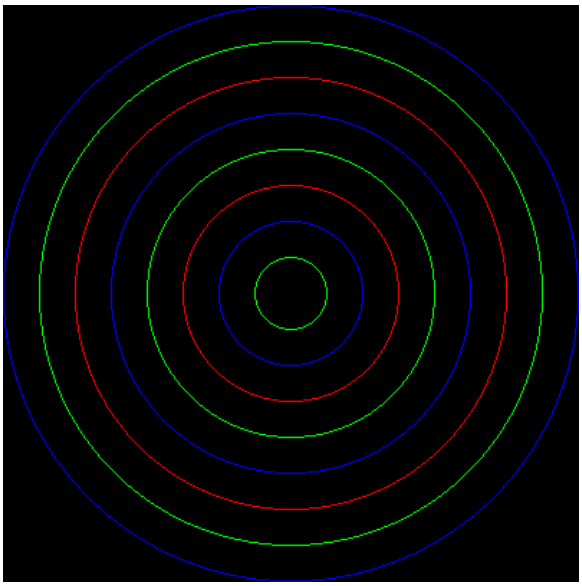

Figure 1: TestCircle Waveform

```
‘include "src/synthesis/math/geometry/Circle.v"
‘include "src/simulation/display/Graphics.v"
‘include "src/simulation/Clock.v"

module TestCircleOctant;

    wire  clk;
    Clock clock(0, clk);

    localparam RADIUS = 200;

    Image #(.width(2*RADIUS+1), .height(2*RADIUS+1)) image();

    reg reset = 1;

    reg signed ‘INT  centerX = RADIUS,
                     centerY = RADIUS;

    reg          ‘INT radius  = RADIUS;
    reg               next    = 1;
    wire signed ‘INT x, y;
    wire              done;

    Circle drawCircle
    (
        clk,
        reset,

        centerX,
        centerY,

        radius,

        next,

        x,
        y,
```

```
        done
    );

    integer file;

    initial
        begin
            //$dumpvars;

            file = $fopen("src/test/math/geometry/Circle/extra/matlabplot.m", "w");

            $fwrite(file, "V=[");
        end

    reg [7:0] R =   0,
              G =   0,
              B = 255;

    always @ (negedge clk)
        begin
            reset <= 0;

            image.setRGB(x, y, R, G, B);

            $fwrite
            (
                   file,
                "[%d, %d];",
                   x, y
            );

            if (done)
                if (radius > 25)
                    begin
                        if (file)
                            begin
                                $fwrite(file, "];\n\n");
                                $fwrite(file, "plot(V(:,1), V(:,2), '.');");
                                $fclose(file);

                                file <= 0;
                            end

                        reset  <= 1;
                        radius <= radius-25;

                        {R, G, B} <= {G, B, R};

                    end
                else
                    begin
                        image.createBMP("src/test/math/geometry/Circle/extra/circles.bmp");

                        #1 $finish;
                    end
            else
                $fwrite(file, "\n");
        end

endmodule
```

6.3 CircleOctant

```
`ifndef INCLUDED_B8B90FE8_1D13_4D77_A6D0_3CF9AC02D968
    `define INCLUDED_B8B90FE8_1D13_4D77_A6D0_3CF9AC02D968
```

```
    `include "src/synthesis/math/number/int/int.v"

    // XST says
    //   It will run at ~94.639 MHz
    //   It is ~139 slices and 4 18x18 multipliers

    // This module produces 1/8 of a circle of a specified radius centered at the origin.

    // Timing:
    // Produces an x,y each cycle after reset is asserted

    module CircleOctant
    (
        input                   clk,
        input                   reset,

        input             `INT radius,

        input                   next,

        output reg signed `INT x,
        output reg signed `INT y
    );

        reg signed `INT p;

        always @ (posedge clk)
            begin
                if (reset)
                    begin
                        // output the first point: (0, radius)
                        x <= 0;
                        y <= radius;

                        // Calculate the initial p "position" for the next cycle.
                        // The following comes from the rounding of integer division from 5/4-radius;
                        // for some reason, the source writes it as (5 - radius*4)/4
                        p <= 1 - radius;
                    end
                else if (next)
                    begin
                        x <= x+1;
                        if (p < 0)
                            p <= p + (x<<1)+3;              // Comes from p += 2*x+1 with x++ before it.
                        else
                            begin
                                y <= y-1;
                                p <= p + ((x-y)<<1)+5;      // Comes from p += 2*(x-y)+1 with the same x++ as
                                                            // above and y-- before it.
                            end
                    end
            end

    endmodule

`endif
//newline required
```

6.3.1 TestCircleOctant

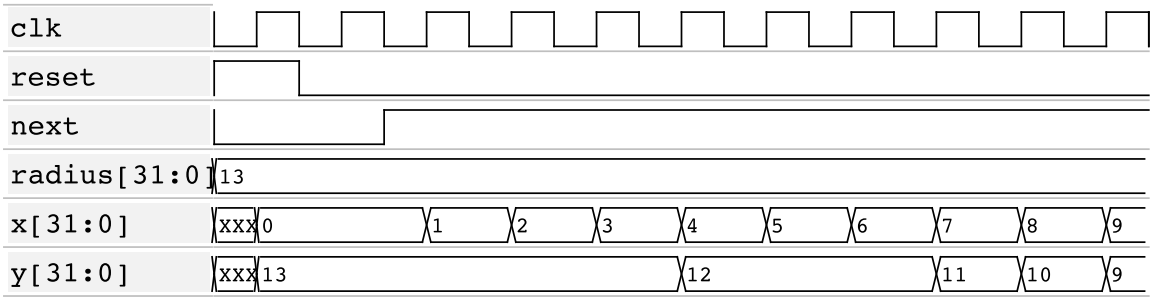

Figure 2: TestCircleOctant Waveform

```
‘include "src/synthesis/math/geometry/CircleOctant.v"
‘include "src/simulation/Clock.v"

module TestCircleOctant;

    wire  clk;
    Clock clock(0, clk);

    reg reset = 1;

    //Works for radius
    //  13
    //  10
    //  5
    //  2

    reg  ‘INT radius = 13;
    reg       next   =  0;
    wire ‘INT x, y;

    CircleOctant octantGenerator
    (
        clk,
        reset,

        radius,

        next,

        x,
        y
    );

    initial
        $dumpvars;

    always @ (negedge clk)
        begin
            $display
            (
                $time,

                "cycle: %d;",
                 clock.cycle,

                " reset: %b;",
                  reset,

                " radius: %d;",
```

```
                radius,

            " next: %b;",
              next,

            " (%d, %d);",
                x,  y
        );

        case (clock.cycle)
              1: reset <= 0;
              2: next  <= 1;
        endcase

        if (x == y)
            #1 $finish;

    end

endmodule
```

7 Display

7.1 DrawerCircle

```
`include "src/synthesis/math/geometry/Circle.v"

module DrawerCircle
#(parameter
    width  = 10'd800,
    height = 10'd600,

    xMax   = width-1,
    yMax   = height-1,

    addrBits = 19,
    dataBits = 36
)(
    input                       clk,
    input                       reset,

    input                       start,

    input     signed `INT       centerX,
    input     signed `INT       centerY,

    input            `INT       radius,

    output  [addrBits-1:0]      address,
    output                      writeDisable,

    output                      finished
);

    wire signed `INT x, y;
    wire             done;

    reg next;

    Circle circle
    (
        clk,
        start || reset,
```

```
        centerX,
        centerY,

        radius,

        next,

        x,y,

        done
    );

    assign finished = done;

    assign address      = x + width*y;
    assign writeDisable = (!next || x < 0 || x > xMax || y < 0 || y > yMax);

    always @ (posedge clk)
        if (reset)
            next <= 0;
        else
            begin
                if (done)
                    next <= 0;

                if (start)
                    next <= 1;
            end

endmodule
```

7.1.1 TestDrawerCircle

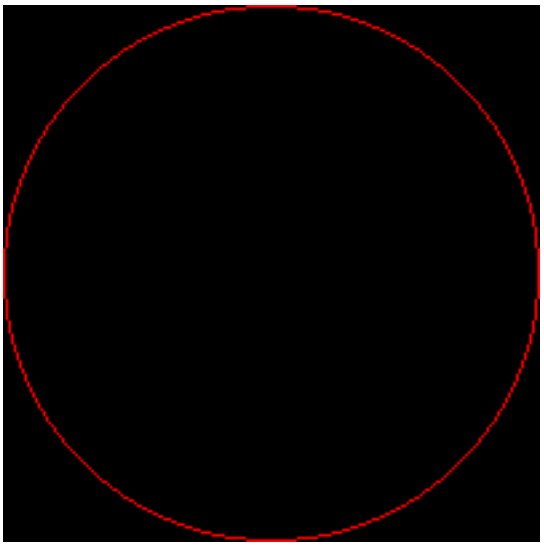

Figure 3:

```
‘include "src/synthesis/display/DrawerCircle.v"
‘include "src/simulation/Clock.v"
‘include "src/simulation/display/Graphics.v"

module TestDrawerCircle;
```

```
    localparam RADIUS = 100,
               SIZE   = 2*RADIUS+1;

    Image #(.width(SIZE), .height(SIZE)) image();

    wire  clk;
    Clock #(.timeClock(5)) clock(0, clk);

    reg reset = 1;

    reg signed  `INT  centerX = RADIUS,
                      centerY = RADIUS;

    reg         `INT  radius  = RADIUS;

    reg               start    = 0;

    wire [18:0]       address;

    wire              writeDisable;
    wire              finished;

    DrawerCircle #(.width(SIZE), .height(SIZE)) drawer
    (
        clk,
        reset,

        start,

        centerX,
        centerY,

        radius,

        address,
        writeDisable,

        finished
    );

    //initial
    //    $dumpvars;

    always @ (posedge clk)
        begin
            //image.setRGB(drawer.x, drawer.y, 0, 0, 255);
            image.data[address] = 24'hFF0000;

            case (clock.cycle)
                1: #3 reset <= 0;
                2: #3 start <= 1;
                3: #3 start <= 0;
            endcase

            if (finished)
                begin
                    image.createBMP("src/test/display/DrawerCircle/extra/circle.bmp");
                    $finish;
                end
        end

endmodule
```

7.2 VGAController

```
////////////////////////////////////////////////////////
////
//// VGAController
//// This module produces the signals necessary to
//// to drive a VGA display.
////
////////////////////////////////////////////////////////

`ifndef INCLUDED_F93C5DED_3CC2_4BA1_AAEB_E6E545A8BE47
    `define INCLUDED_F93C5DED_3CC2_4BA1_AAEB_E6E545A8BE47

    module VGAController

    ////////////////////
    //// Parameters ////
    ////////////////////

    #(parameter

        hActive      = 640,     // Active screen width  in pixels.
        hPorchFront  = 16,      // Horizontal front porch length in pixels (x).
        hSync        = 96,      // Horizontal sync        length in pixels (x).
        hPorchBack   = 48,      // Horizontal back  porch length in pixels (x).
        hBits        = 10,      // Bits required to count the number
                                //     of pixels in a line, including
                                //     the blanking region.

        vActive      = 480,     // Active screen height in pixels.
        vPorchFront  = 11,      // Vertical front porch length in lines (y).
        vSync        = 2,       // Vertical sync        length in lines (y).
        vPorchBack   = 31,      // Vertical back  porch length in lines (y).
        vBits        = 10       // Bits required to count the number
                                //     of lines in an image, including
                                //     the blanking region.
    )

    ///////////////
    //// Ports ////
    ///////////////

    (
        input  clk,                 // HIGH: trigger logic; the pixel clock.
        input  reset,               // HIGH: asynchronous reset.

        output reg blank,           // LOW: in blanking region
        output reg syncH,           // LOW: in horizontal syncing region.
        output reg syncV,           // LOW: in vertical   syncing region.

                                    // The origin is in the
                                    // top left of the screen.
        output reg [hBits-1:0] x,   // Current pixel in  a  line  ; x-coordinate
        output reg [vBits-1:0] y    // Current line  in the image ; y-coordinate
    );

        //////////////
        //// Body ////
        //////////////

        localparam [hBits-1:0] hActiveEnd     = hActive-1,
                               hPorchFrontEnd = hActiveEnd     + hPorchFront,
                               hSyncEnd       = hPorchFrontEnd + hSync,
                               hPorchBackEnd  = hSyncEnd       + hPorchBack;

        localparam [vBits-1:0] vActiveEnd     = vActive-1,
                               vPorchFrontEnd = vActiveEnd     + vPorchFront,
                               vSyncEnd       = vPorchFrontEnd + vSync,
                               vPorchBackEnd  = vSyncEnd       + vPorchBack;
```

```
        reg vIsActive; // We need to keep state for handling blank.

        always @ (posedge clk or posedge reset)
            if (reset)
                begin
                    vIsActive <= 1;

                    blank <= 1;

                    syncH <= 1;
                    syncV <= 1;

                    x <= 0;
                    y <= 0;
                end
            else
                begin
                    x <= x+1;

                    case (x)
                        default:;
                        hActiveEnd     : blank <= 0;  // Screen ends this cycle.
                        hPorchFrontEnd : syncH <= 0;  // Front Porch ends this cycle.
                        hSyncEnd       : syncH <= 1;  // Sync ends this cycle
                        hPorchBackEnd  :              // Back porch (line) ends this cycle.
                            begin
                                x <= 0;
                                y <= y+1;

                                if (vIsActive)        // We above vertical blanking?
                                    blank <= 1;

                                case (y)              // Handle the end of special lines.
                                    default:;
                                    vActiveEnd:                   // Active region (screen) ends.
                                        begin
                                            vIsActive <= 0;       // Screen no longer active.
                                            blank     <= 0;
                                        end

                                    vPorchFrontEnd : syncV <= 0;  // Front Porch ends.
                                    vSyncEnd       : syncV <= 1;  // Sync ends.

                                    vPorchBackEnd:                // Back Porch (vertical scan) ends.
                                        begin
                                            vIsActive <= 1;       // Begin active region again.
                                            blank     <= 1;
                                            y         <= 0;
                                        end
                                endcase
                            end
                    endcase
                end
    endmodule

`endif
//newline required
```

8 Memory

8.1 ZBT

```
`ifndef INCLUDE_EF64810C_BC96_417C_848A_5C61421DEE40
```

```
`define INCLUDE_EF64810C_BC96_417C_848A_5C61421DEE40

`include "src/synthesis/util/Delay.v"

module ZBT
(
    ////////////////////////
    //// User Interface ////
    ////////////////////////

    input         clk,                // Assumed to be in phase with system clk.
    input         reset,

    input         read,               // LOW: write; HIGH: read

    input  [18:0] address,
    input  [35:0] dataToWrite,
    output [35:0] dataToRead,

    ////////////////////////
    //// Control Signals ////
    ////////////////////////

    output [18:0] ramAddress,          // address
    inout  [35:0] ramData,             // data
    output        ramDisable,          // ce_b
    output        ramDisableOutput,    // oe_b
    output        ramDisableWrite,     // we_b
    output [3:0]  ramDisableWriteByte, // bwe_b
    output        ramDisableClock,     // cen_b
    output        ramEnableBurstMode,  // adv_ld
    output        ramClock             // clk
);

    // The data to write must be presented
    // to the ZBT 2 cycles after a write is
    // is requested; the request is carried
    // along with the data, so that the inout
    // (tristate bus) can be properly multiplexed.

    wire        readDelayed;
    wire [35:0] dataToWriteDelayed;

    Delay #(.delay(2)) delayRead(clk, read, readDelayed); // separately to infer dedicated hardware.
    Delay #(.delay(2), .width(36)) delayDataToWrite(clk, dataToWrite, dataToWriteDelayed);

    reg        readDelayedMore;         // Make sure there is no contention.
    reg [35:0] dataToWriteDelayedMore;

    always @ (negedge clk or posedge reset)
        if (reset)
            begin
                readDelayedMore        <= 1;
                dataToWriteDelayedMore <= 0;
            end
        else
            begin
                readDelayedMore        <= readDelayed;
                dataToWriteDelayedMore <= dataToWriteDelayed;
            end

    //////////////////////
    // Assign Controls //
    //////////////////////

    assign ramAddress          = address,
           ramData             = readDelayedMore ? 36'hz : dataToWriteDelayedMore,
           ramDisable          = 0,
           ramDisableOutput    = 0,
```

```
                ramDisableWrite     = read,
                ramDisableWriteByte = 0,
                ramDisableClock     = 0,
                ramEnableBurstMode  = 0,
                ramClock            = clk,
                dataToRead          = ramData;

    endmodule

`endif
//newline required
```

8.1.1 TestZBT

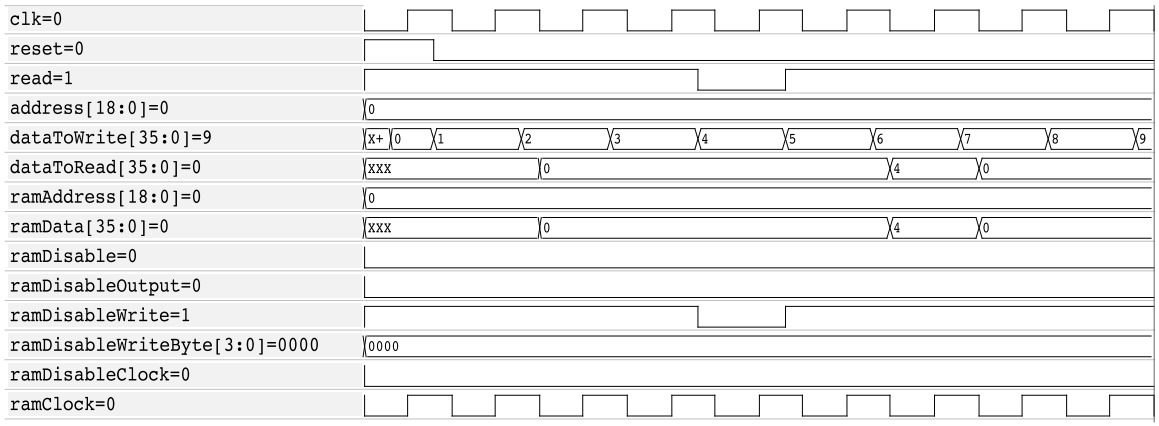

Figure 4: TestZBT Waveform

```
`include "src/synthesis/memory/ZBT.v"
`include "src/simulation/Clock.v"

module TestZBT;

    wire clk;
    Clock #(.timeClock(5)) clock(0, clk);

    reg            reset       = 1;

    reg            read        = 1;
    reg  [18:0]    address     = 0;
    wire [35:0] #3 dataToWrite = clock.cycle;
    wire [35:0]    dataToRead;

    wire [18:0] ramAddress;
    wire [35:0] ramData     = zbt.readDelayedMore ? 36'b0 : 36'hz;
    wire        ramDisable;
    wire        ramDisableOutput;
    wire        ramDisableWrite;
    wire [3:0]  ramDisableWriteByte;
    wire        ramDisableClock;
    wire        ramEnableBurstMode;
    wire        ramClock;

    ZBT zbt
    (
        clk,
        reset,
```

```
        read,

        address,
        dataToWrite,
        dataToRead,

        ramAddress,
        ramData,
        ramDisable,          // ce_b
        ramDisableOutput,    // oe_b
        ramDisableWrite,     // we_b
        ramDisableWriteByte, // bwe_b
        ramDisableClock,     // cen_b
        ramEnableBurstMode,  // adv_ld
        ramClock
    );

    initial
        $dumpvars;

    always @ (posedge clk)
        case (clock.cycle)
             1: #3 reset <= 0;
             4: #3 read  <= 0;
             5: #3 read  <= 1;
            10: $finish;
        endcase

endmodule
```

8.2 BufferSwitcher

```
`ifndef INCLUDE_69037244_E66C_45C2_8890_680FCB7C4BFC
    `define INCLUDE_69037244_E66C_45C2_8890_680FCB7C4BFC

    module BufferSwitcher
    #(parameter
        addrBits = 19,
        dataBits = 36
    )(
        ////////////////////////
        //// User Interface ////
        ////////////////////////

        input                  clk,               // Must be >= clkA, clkB
        input                  reset,

        input                  clkA,
        input                  clkB,

        input                  finishedA,    // Must be kept high until it's notified of switching
        input                  finishedB,    // Must be kept high until it's notified of switching

        input                  readA,
        input                  readB,

        input  [addrBits-1:0] addressA,
        input  [addrBits-1:0] addressB,

        input  [dataBits-1:0] dataToWriteA,
        input  [dataBits-1:0] dataToWriteB,

        output [dataBits-1:0] dataToReadA,
        output [dataBits-1:0] dataToReadB,

        output reg            finishedSwitchingA,
```

```
    output reg            finishedSwitchingB,

    ////////////////////////
    //// Control Signals ////
    ////////////////////////

    output                clk0,
    output                clk1,

    output                read0,
    output                read1,

    output [addrBits-1:0] address0,
    output [addrBits-1:0] address1,

    output [dataBits-1:0]  dataToWrite0,
    output [dataBits-1:0]  dataToWrite1,

    input  [dataBits-1:0]  dataToRead0,
    input  [dataBits-1:0]  dataToRead1
);

reg connect1ToA;
reg finishedSwitching;

reg finishedAWentLow,
    finishedBWentLow;

always @ (posedge clk or posedge reset)
    if (reset)
        begin
            connect1ToA       <= 0;
            finishedSwitching <= 0;

            finishedAWentLow  <= 0;
            finishedBWentLow  <= 0;
        end
    else if (finishedA && finishedB)
        begin
            if (!finishedSwitching)
                connect1ToA <= !connect1ToA;

            finishedSwitching <= 1;
        end
    else if (finishedSwitching)
        if (!(finishedA || finishedB) || (finishedAWentLow && finishedBWentLow))
            begin
                finishedSwitching      <= 0;
                finishedAWentLow <= 0;
                finishedBWentLow <= 0;
            end
        else if (finishedA)
            finishedBWentLow <= 1;
        else if (finishedB)
            finishedAWentLow <= 1;

reg finishedSwitchingASent;
always @ (posedge clkA or posedge reset)
    if (reset)
        begin
            finishedSwitchingASent <= 0;
            finishedSwitchingA     <= 0;
        end
    else
        begin
            finishedSwitchingA <= 0;

            if (finishedSwitching && !finishedSwitchingASent)
                begin
```

```
                    finishedSwitchingA     <= 1;
                    finishedSwitchingASent <= 1;
                end

            if (finishedSwitchingASent && !finishedSwitching)
                finishedSwitchingASent <= 0;
        end

    reg finishedSwitchingBSent;
    always @ (posedge clkB or posedge reset)
        if (reset)
            begin
                finishedSwitchingBSent <= 0;
                finishedSwitchingB     <= 0;
            end
        else
            begin
                finishedSwitchingB <= 0;

                if (finishedSwitching && !finishedSwitchingBSent)
                    begin
                        finishedSwitchingB     <= 1;
                        finishedSwitchingBSent <= 1;
                    end

                if (finishedSwitchingBSent && !finishedSwitching)
                    finishedSwitchingBSent <= 0;
            end

    assign  clk0         = connect1ToA ? clkB         : clkA         ,
            clk1         = connect1ToA ? clkA         : clkB         ,
            read0        = connect1ToA ? readB        : readA        ,
            read1        = connect1ToA ? readA        : readB        ,
            address0     = connect1ToA ? addressB     : addressA     ,
            address1     = connect1ToA ? addressA     : addressB     ,
            dataToWrite0 = connect1ToA ? dataToWriteB : dataToWriteA ,
            dataToWrite1 = connect1ToA ? dataToWriteA : dataToWriteB ,

            dataToReadA  = connect1ToA ? dataToRead1  : dataToRead0  ,
            dataToReadB  = connect1ToA ? dataToRead0  : dataToRead1  ;

    endmodule

`endif
//newline required
```

8.2.1 TestBufferSwitcher

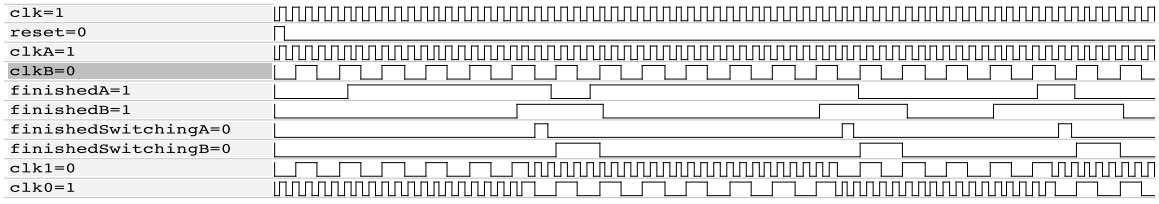

Figure 5: TestBufferSwitcher Waveform

```
`include "src/synthesis/memory/BufferSwitcher.v"
`include "src/simulation/Clock.v"

module TestZBT;
```

```
wire clk;
Clock #(.timeClock(5))  clock(0, clk);

wire clkA;
Clock #(.timeClock(5))  clockA(0, clkA);

wire clkB;
Clock #(.timeClock(17)) clockB(0, clkB);

reg            reset            = 1;

reg            finishedA        = 0,
               finishedB        = 0;

reg            readA            = 0,
               readB            = 0;

reg  [18:0]    addressA         = 0,
               addressB         = 0;

reg  [35:0]    dataToWriteA     = 0,
               dataToWriteB     = 0;

wire [35:0]    dataToReadA,
               dataToReadB;

wire        read0,read1;
wire [18:0] address0, address1;

wire [35:0] dataToWrite0 , dataToWrite1,
            dataToRead0  , dataToRead1;

wire        finishedSwitchingA,
            finishedSwitchingB;

BufferSwitcher bufferSwitcher
(
    clk,
    reset,

    clkA,
    clkB,

    finishedA,
    finishedB,

    readA,
    readB,

    addressA,
    addressB,

    dataToWriteA,
    dataToWriteB,

    dataToReadA,
    dataToReadB,

    finishedSwitchingA,
    finishedSwitchingB,

    clk0,
    clk1,

    read0,
    read1,

    address0,
```

```
        address1,

        dataToWrite0,
        dataToWrite1,

        dataToRead0,
        dataToRead1
    );

    initial
        $dumpvars;

    always @ (posedge clk)
        case (clock.cycle)
              1: #3 reset <= 0;
             70: $finish;
        endcase

    always @ (posedge clkA)
        begin
            case (clockA.cycle)
                6,25,60: #3 finishedA <= 1;
            endcase

            if (finishedSwitchingA)
                #3 finishedA <= 0;
        end

    always @ (posedge clkB)
        begin
            case (clockB.cycle)
                6,13,17: #3 finishedB <= 1;
            endcase

            if (finishedSwitchingB)
                #3 finishedB <= 0;
        end
endmodule
```

9 Control

9.1 InteractionScheduler

```
‘ifndef INCLUDE_5DDC94CF_706A_4049_9A5F_009606AFDC6F
    ‘define INCLUDE_5DDC94CF_706A_4049_9A5F_009606AFDC6F

    module InteractionScheduler
    #(parameter
        objects               = 512,
        objectsBits           = 9,
        accumulatorStages     = 13
    )(
        input                         clk,
        input                         reset,

        output reg [objectsBits-1:0] primary,
        output reg [objectsBits-1:0] secondary,

        output reg                    finished
    );

        localparam STAGE_MAX        = objects-1,
                   SUBSTAGES        = objects/2,
```

```
            SUBSTAGE_MAX     = SUBSTAGES-1,
            BUBBLE_SECONDARY = STAGE_MAX,
            JUMP_SECONDARY   = STAGE_MAX-1;

reg [objectsBits-1:0] stage;

reg stageFirst,
    stageOdd;

reg [objectsBits-2:0]  substage,
                       substageJumpEven,
                       substageJumpOdd,
                       substageBubbleEven,
                       substageBubbleOdd;

reg [objectsBits-1:0]  primaryNext,
                       secondaryNext,

                       jumpPrimary,

                       bubbleEvenPrimary,
                       bubbleOddPrimary;

wire [objectsBits-1:0] stageNext = stage-1;

always @ (posedge clk)
    if (reset)
        begin
            finished           <= 0;

            stage              <= STAGE_MAX;
            stageFirst         <= 1;
            stageOdd           <= 1;

            substage           <= 0;
            substageJumpEven   <= SUBSTAGE_MAX;
            substageJumpOdd    <= SUBSTAGE_MAX-1;
            substageBubbleEven <= SUBSTAGE_MAX-(accumulatorStages-1);
            substageBubbleOdd  <= SUBSTAGE_MAX;

            primary            <= 0;
            secondary          <= STAGE_MAX;

            primaryNext        <= 1;
            secondaryNext      <= STAGE_MAX-1;

            jumpPrimary        <=  objects-3;    // from (objects-4)+(objects-1)-(objects-2)

            bubbleEvenPrimary  <= (objects-2)/2;
            bubbleOddPrimary   <=  objects-2;    // from (STAGE_MAX + currentStage)/2 = ((objects-1)+(objects-3))/2
        end
    else
        if (substage == SUBSTAGE_MAX)
            if (stage == 1)
                finished <= 1;
            else
                begin
                    stageFirst    <= 0;
                    stageOdd      <= !stageOdd;
                    stage         <= stageNext;
                    substage      <= 0;

                    primary       <= 0;
                    secondary     <= stageNext;

                    primaryNext   <= 1;
                    secondaryNext <= stageNext-1;
                end
        else
```

```
                    begin
                        if ((substage == substageBubbleEven-1) && !stageOdd && !stageFirst)
                            begin
                                secondary         <= BUBBLE_SECONDARY;
                                primary           <= bubbleEvenPrimary;
                                bubbleEvenPrimary <= bubbleEvenPrimary-1;
                            end
                        else if ((substage == substageBubbleOdd-1) && stageOdd && !stageFirst)
                            begin
                                secondary         <= BUBBLE_SECONDARY;
                                primary           <= bubbleOddPrimary;
                                bubbleOddPrimary  <= bubbleOddPrimary-1;
                            end
                        else if (((!stageOdd && (substage == substageJumpEven-1)) ||
                                  (stageOdd && (substage == substageJumpOdd-1)))
                                            && (stage < STAGE_MAX-2))
                            begin
                                primary       <= jumpPrimary;
                                secondary     <= JUMP_SECONDARY;

                                primaryNext   <= jumpPrimary+1;
                                secondaryNext <= JUMP_SECONDARY-1;

                                jumpPrimary   <= jumpPrimary-1;

                                if (stageOdd)
                                    begin
                                        if (substageJumpEven == substageBubbleEven+1)
                                            substageJumpEven <= substageJumpEven-2;
                                        else
                                            substageJumpEven <= substageJumpEven-1;

                                        substageJumpOdd <= substageJumpOdd-1;
                                    end
                            end
                        else
                            begin
                                primary       <= primaryNext;
                                secondary     <= secondaryNext;
                                primaryNext   <= primaryNext+1;
                                secondaryNext <= secondaryNext-1;
                            end

                        substage <= substage+1;
                    end
    endmodule
`endif
//newline required
```

9.1.1 TestInteractionScheduler

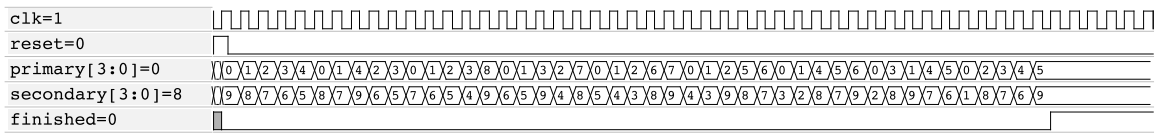

Figure 6: TestInteractionScheduler Waveform

```
`ifdef HOME
    `include "src/synthesis/control/InteractionScheduler.v"
    `include "src/simulation/Clock.v"
`endif
```

```
module TestInteractionScheduler;

    wire clk;
    Clock #(.timeClock(5)) clock(0, clk);

    localparam OBJECTS      = 10,               // If you make these bigger,
               OBJECTS_BITS =  4,               // you should probably remove
               ACCUM_STAGES =  3;               // the $dumpvars below.

    wire [OBJECTS_BITS-1:0] primary,
                            secondary;

    reg  reset = 1;
    wire finished;

    InteractionScheduler
    #(
        .objects(OBJECTS),
        .objectsBits(OBJECTS_BITS),
        .accumulatorStages(ACCUM_STAGES)
    )
    scheduler
    (
        clk,
        reset,

        primary,
        secondary,

        finished
    );

    initial
        $dumpvars;

    always @ (posedge clk)
        begin
            case (clock.cycle)
                 1: #3 reset <= 0;
            endcase

            if (finished)
                #50 $finish;
        end

    always @ (negedge clk)
        $display("(%d, %d)", primary, secondary);

endmodule
```

9.2 InteractionScheduler.py

10 Physics

10.1 GravitationalAccelerator

```
‘ifndef INCLUDED_F460DB7C_3257_45ED_90ED_E6C400BD991F
    ‘define INCLUDED_F460DB7C_3257_45ED_90ED_E6C400BD991F

    ‘include "src/synthesis/math/number/real/RealSqrt.v"
```

```
`include "src/synthesis/math/number/real/RealDiv.v"

`include "src/synthesis/math/vector/VectorSub.v"
`include "src/synthesis/math/vector/VectorDot.v"
`include "src/synthesis/math/vector/VectorDivReal.v"
`include "src/synthesis/math/vector/VectorNeg.v"
`include "src/synthesis/math/vector/VectorMulReal.v"

`include "src/synthesis/util/Delay.v"

`define GravitationalAccelerator_DELAY_OBJ1     \
 (                                              \
      `VectorSub_DELAY                          \
     +`VectorDot_DELAY                          \
     +`RealSqrt_DELAY                           \
     +`VectorDivReal_DELAY                      \
     +`VectorMulReal_DELAY                      \
 )

`define GravitationalAccelerator_DELAY_OBJ2     \
 (                                              \
     `GravitationalAccelerator_DELAY_OBJ1       \
    +`VectorNeg_DELAY                           \
 )

//////////////////
////Discussion////
//////////////////

// Xilinx says this will run at about 133 Mhz.

module GravitationalAccelerator
(
    input          clk,

    input  `VECTOR obj1Center,
    input  `REAL   obj1GMass,

    input  `VECTOR obj2Center,
    input  `REAL   obj2GMass,

    output `VECTOR obj1Accel,
    output `VECTOR obj2Accel
);

    wire `REAL   obj1GMassDelayed,
                 obj2GMassDelayed,

                 distanceSquared,
                 distance,

                 obj1AccelMag,
                 obj1AccelMagDelayed,

                 obj2AccelMag,
                 obj2AccelMagDelayed;

    wire `VECTOR vectorFrom1To2,
                 vectorFrom1To2Delayed,

                 obj1AccelDir,
                 obj2AccelDir;


    ////////////
    //// #1 ////
    ////////////

    VectorSub getVectorFrom1To2(clk, obj2Center, obj1Center, vectorFrom1To2);
```

```
        ////////////
        //// #2 ////
        ////////////

        Delay #(.delay(`VectorSub_DELAY+`VectorDot_DELAY), .width(`REAL_BITS))
                delayObj1GMass(clk, obj1GMass, obj1GMassDelayed);

        Delay #(.delay(`VectorSub_DELAY+`VectorDot_DELAY), .width(`REAL_BITS))
                delayObj2GMass(clk, obj2GMass, obj2GMassDelayed);

        VectorDot getDistanceSquared(clk, vectorFrom1To2, vectorFrom1To2, distanceSquared);

        ////////////
        //// #3 ////
        ////////////

        RealSqrt getDistance(clk, distanceSquared , distance);

        RealDiv  getObj1AccelMag(clk , obj2GMassDelayed, distanceSquared, obj1AccelMag);
        RealDiv  getObj2AccelMag(clk , obj1GMassDelayed, distanceSquared, obj2AccelMag);

        Delay #(.delay(`VectorDot_DELAY+`RealSqrt_DELAY), .width(`VECTOR_BITS))
                delayVectorFrom1To2(clk, vectorFrom1To2, vectorFrom1To2Delayed);

        ////////////
        //// #4 ////
        ////////////

        VectorDivReal getObj1AccelDir(clk, vectorFrom1To2Delayed, distance, obj1AccelDir);

        Delay #(.delay(`RealSqrt_DELAY+`VectorDivReal_DELAY-`RealDiv_DELAY), .width(`REAL_BITS))
                delayObj1AccelMag(clk, obj1AccelMag, obj1AccelMagDelayed);

        ////////////
        //// #5 ////
        ////////////

        VectorNeg getObj2AccelDir(clk, obj1AccelDir, obj2AccelDir);

        VectorMulReal getObj1Accel(clk, obj1AccelDir, obj1AccelMagDelayed, obj1Accel);

        Delay #(.delay(`RealSqrt_DELAY+`VectorDivReal_DELAY+`VectorNeg_DELAY-`RealDiv_DELAY), .width(`REAL_BITS))
                delayObj2AccelMag(clk, obj2AccelMag, obj2AccelMagDelayed);

        ////////////
        //// #6 ////
        ////////////

        VectorMulReal getObj2Accel(clk, obj2AccelDir, obj2AccelMagDelayed, obj2Accel);

    endmodule

`endif
```

10.1.1 TestGravitationalAccelerator

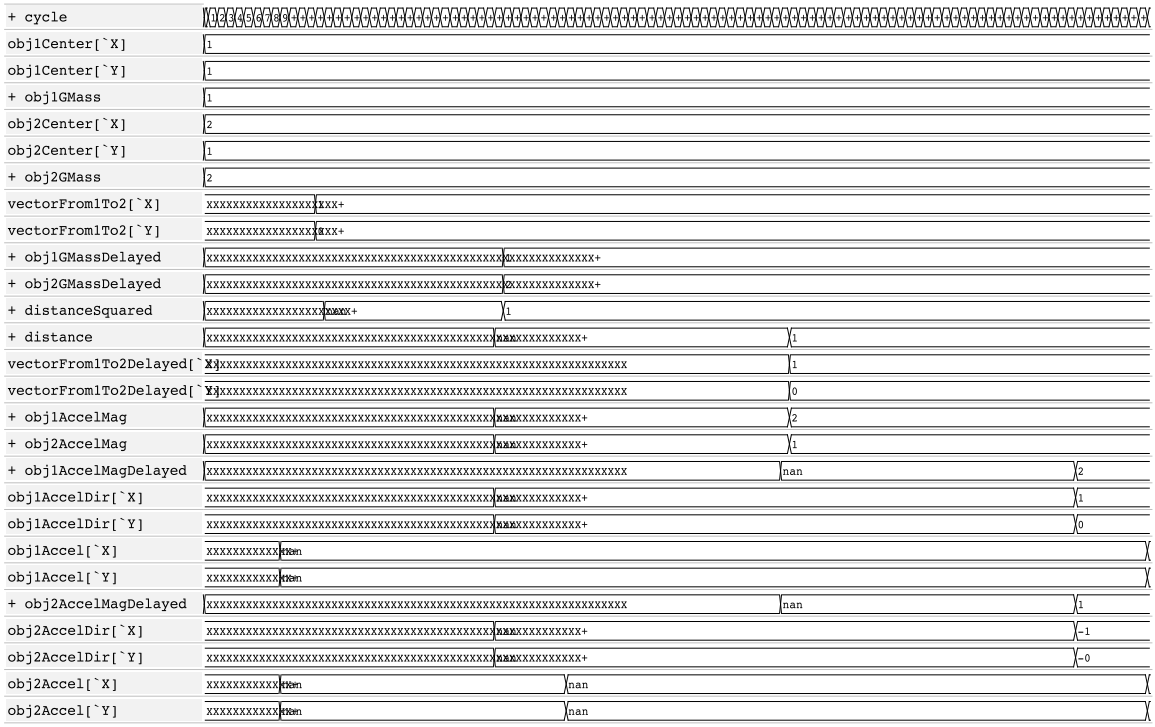

Figure 7: TestGravitationalAccelerator Waveform

```
‘ifdef HOME
    ‘include "src/synthesis/physics/GravitationalAccelerator.v"
    ‘include "src/simulation/Clock.v"
‘endif

module TestGravitationalAccelerator;

    wire  clk;
    Clock clock(0, clk);

    wire ‘VECTOR obj1Accel, obj2Accel;

    GravitationalAccelerator accelerator
    (
        clk,

‘ifdef REAL_FIXED
        //do something
‘else
        {$realtobits(1.0),$realtobits(1.0)},
        $realtobits(1.0),

        {$realtobits(2.0),$realtobits(1.0)},
        $realtobits(2.0),
‘endif

        obj1Accel,
        obj2Accel
    );

    initial
        $dumpvars;
```

```
    always @ (posedge clk)
        begin
            $display
            (
                "cycle %d"  , clock.cycle,
`ifdef REAL_FIXED
                 //do something
`else
                ": (%f, %f)", $bitstoreal(obj1Accel[`X]), $bitstoreal(obj1Accel[`Y]),
                "\t(%f, %f)", $bitstoreal(obj2Accel[`X]), $bitstoreal(obj2Accel[`Y])
`endif
            );

            if (clock.cycle == `GravitationalAccelerator_DELAY_OBJ2)
                $display("\n//// Should be: (2.000000, 0.000000) (-1.000000, 0.000000) //// \n");

            if (clock.cycle == `GravitationalAccelerator_DELAY_OBJ2+1)
                $finish;
        end

endmodule
```

10.2 Constants

```
`ifndef INCLUDED_C6587C25_67DD_4F5A_B167_52EE01D80EE1

    `define INCLUDED_C6587C25_67DD_4F5A_B167_52EE01D80EE1

    `ifdef XILINX_TOOLSET
    `else
        `include "../math/vector/vector.v"
    `endif

    `define GRAVITATIONAL <someNumberHere>

`endif
```

11 User Interface: PS/2 Mouse

11.1 PS2Mouse

```
`ifndef INCLUDE_F6603C45_F426_4DB0_B177_D3E7E7E6CE0D
    `define INCLUDE_F6603C45_F426_4DB0_B177_D3E7E7E6CE0D

    `include "src/synthesis/io/ps2/PS2MouseController.v"
    `include "src/synthesis/io/ps2/PS2Receiver.v"
    `include "src/synthesis/io/ps2/PS2Sender.v"
    `include "src/synthesis/io/ps2/PS2MouseDecoder.v"
    `include "src/synthesis/util/LevelToPulseNeg.v"
    `include "src/synthesis/util/FIFO.v"

    module PS2Mouse
    #(parameter
        xMax   = 799,
        xBits  = 10,
        xStart = 394,

        yMax   = 599,
```

```
    yBits  = 10,
    yStart = 295
)(
    input                       clk,
    input                       reset,

    inout                       mouseClock,
    inout                       mouseData,

    input                       next,

    output reg  [xBits-1:0]       x,
    output reg  [yBits-1:0]       y,
    output      [2:0]           buttons,

    output                      bufferFull,
    output                      dataAvailable
);

    ////////////////////
    //// Controller ////
    ////////////////////

    wire [7:0] dataToReceive,
               dataToSend;

    wire       send,
               read,
               timerExpired,
               sendFinished,
               byteAvailable,
               decoderEnable;

    PS2MouseController controller
    (
        clk,
        reset,

        byteAvailable,
        sendFinished,

        dataToReceive,
        dataToSend,

        timerExpired,

        read,
        send,

        decoderEnable
    );

    ////////////////////////
    //// Mouse Clock ////
    ////////////////////////

    // The system may sample mouseClock
    // when it's in transition. Consequently,
    // it is necessary to wait for metastability
    // to settle.

    wire mouseClockStable;
    Delay #(.delay(2)) killMetaStability
    (
        clk,
        mouseClock,
        mouseClockStable
    );
```

```
wire mouseClockWentLow;
LevelToPulseNeg trackMouseClock
(
    clk,

    reset, 0,

    mouseClockStable,
    mouseClockWentLow
);

//////////////////////
//// PS2 Interface ////
//////////////////////

PS2Sender sender
(
    clk, reset,

    mouseClock,
    mouseClockWentLow,
    mouseData,

    dataToSend,
    send,

    timerExpired,

    sendFinished
);

wire    dataRead;

PS2Receiver receiver
(
    clk, reset,

    mouseClock,
    mouseClockWentLow,
    mouseData,

    read,

    byteAvailable,
    dataToReceive
);

///////////////
//// Mouse ////
///////////////
wire dataDecodedAvailable;

wire [2:0] buttonsDecoded;

wire signed [8:0] xDelta,
                  yDelta;

PS2MouseDecoder decoder
(
    clk, reset,

    decoderEnable,

    dataToReceive,
    byteAvailable,

    xDelta, yDelta,

    buttonsDecoded,
```

```
            dataDecodedAvailable
        );

        //////////////
        //// FIFO ////
        //////////////

        wire full;

        wire signed [8:0] xDeltaBuffered,
                          yDeltaBuffered;

        FIFO fifo
        (
            clk, reset,

            {xDelta, yDelta, buttonsDecoded},
            dataDecodedAvailable,
            next,

            bufferFull,
            dataAvailable,

            {xDeltaBuffered, yDeltaBuffered, buttons}
        );

        ///////////////
        //// Logic ////
        ///////////////

        wire signed [xBits:0] xNext = x + xDeltaBuffered;
        wire signed [yBits:0] yNext = y - yDeltaBuffered;  // Screen coordinates: deltaY increases downward.

        always @ (posedge clk)
            if (reset)
                begin
                    x <= xStart;
                    y <= yStart;
                end
            else if (next && dataAvailable)
                begin
                    x <= (xNext > xMax) ? xMax : (xNext < 0) ? 0 : xNext;
                    y <= (yNext > yMax) ? yMax : (yNext < 0) ? 0 : yNext;
                end

    endmodule

`endif
// newline required
```

11.1.1 TestPS2Mouse

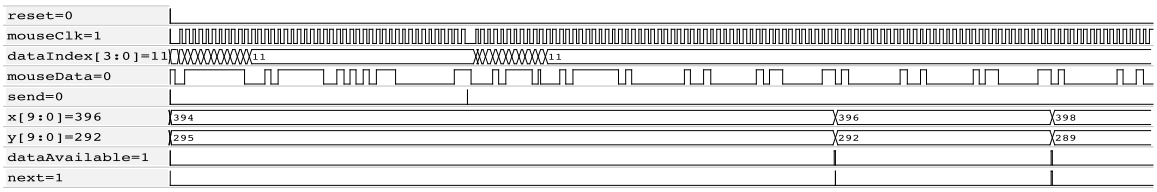

Figure 8: TestPS2Mouse Waveform

```
`include "src/synthesis/io/ps2/PS2Mouse.v"
`include "src/simulation/Clock.v"

module TestPS2Mouse;

    wire clk;
    Clock #(.timeClock(10)) clock(0, clk);

    reg  mouseClockEnable = 0;
    wire clkMouse;

    Clock #(.timeClock(100)) mouseClock(!mouseClockEnable, clkMouse);

    reg mouseDataReg = 1;

    //iverilog sucks
    wire mouseClkTemp  = mouseClockEnable ? clkMouse     : 1;

    wire (pull1, strong0) mouseClk  = mouseClkTemp;
    wire (pull1, strong0) mouseData = mouseDataReg;

    reg reset = 1,
        next  = 0;

    wire [9:0]  x, y;
    wire [2:0]  buttons;

    wire        bufferFull,
                dataAvailable;

    PS2Mouse mouse
    (
        clk,
        reset,

        mouseClk,
        mouseData,

        next,

        x,
        y,
        buttons,

        bufferFull,
        dataAvailable
    );

    //// Change Parameters ////

    defparam mouse.controller.timerTimeClock  = 2,
             mouse.controller.timerTimeData   = 2,
             mouse.controller.timerTimePacket = 1023,
             mouse.controller.timerTimeBits   = 10,
             mouse.controller.enablerBits     = 2,
             mouse.controller.enablerCycles   = 2;

    initial
        $dumpvars;

    always @ (posedge clk)
        begin
            if (mouse.timerExpired)
                begin
                    #3 mouseClockEnable  <= 1;
                end

            next <= 0; // pulse
```

```
            if (dataAvailable && !next)
                next <= 1;

            case (clock.cycle)
                 1: #3 reset <= 0;

                1500: #3 $finish;
            endcase
        end

    //// Act Like Mouse ////

    reg [31:0] dataToSend;

    always @ (mouse.controller.state)
        case (mouse.controller.state)
             3: dataToSend <= {23'b0, 9'b1111_1010_0};
             4: dataToSend <= {23'b0, 9'b1010_1010_0};
             5: dataToSend <= {23'b0, 9'b0000_0000_0};
             7: dataToSend <= {23'b0, 9'b1111_1010_0};
             8: dataToSend <= 32'b100000011_0_1_000000010_0_1_000000001_0;
        endcase

    integer bitIndex = 0;

    always @ (posedge mouseClk)
        begin

            case (mouse.sender.dataIndex)       // Ack Bit
                10: #3 mouseDataReg <= 0;
                11: #3 mouseDataReg <= 1;
            endcase

            case (mouse.controller.state)
                3,4,5,7: begin
                             bitIndex     <= (bitIndex < 10) ? bitIndex+1          : 0;
                             mouseDataReg <= (bitIndex < 9) ? dataToSend[bitIndex] : 1;
                         end

                       8: begin
                             bitIndex     <= (bitIndex < 32) ? bitIndex+1           : 0;
                             mouseDataReg <= (bitIndex < 32) ? dataToSend[bitIndex] : 1;
                         end
            endcase
        end

endmodule
```

11.2 PS2MouseController

```
`ifndef INCLUDE_17D3B7CF_4DC1_4632_8F30_D984407D1D26
    `define INCLUDE_17D3B7CF_4DC1_4632_8F30_D984407D1D26

    `include "src/synthesis/util/Enabler.v"
    `include "src/synthesis/util/EnableCounter.v"

    module PS2MouseController
    (
        input               clk,
        input               reset,

        input               dataAvailable,
        input               sendFinished,

        input      [7:0]    dataToReceive,
        output reg [7:0]    dataToSend,
```

```
    output              timerExpired,

    output reg          read,
    output reg          send,

    output reg          decoderEnable
);

    ///////////////
    //// Timer ////
    ///////////////

    parameter timerTimeClock  =  100,
              timerTimeData   =    5,
              timerTimePacket = 2000,
              timerTimeBits   =   11;

    reg [timerTimeBits-1:0] timerTime;
    reg        timerStart;
    wire       enableOneMicrosecond;

    Enabler #(.countBits(6), .cyclesPerEnable(40)) enablerOneMicrosecond
    (
        clk,
        timerStart,
        enableOneMicrosecond
    );

    EnableCounter #(.countBits(timerTimeBits)) timer
    (
        clk,
        timerStart,
        enableOneMicrosecond,
        timerTime,
        timerExpired
    );

    ///////////////
    //// Logic ////
    ///////////////

    localparam CODE_RESET         = 8'hFF,
               CODE_ACK           = 8'hFA,
               CODE_ACK_BAT       = 8'hAA,
               CODE_ID            = 8'h00,
               CODE_ENABLE        = 8'hF4;

    localparam SEND_TIMER_1       = 4'd0,
               SEND_TIMER_2       = 4'd1,
               SEND_RESET         = 4'd2,
               RECEIVE_ACK_RESET  = 4'd3,
               RECEIVE_ACK_BAT    = 4'd4,
               RECEIVE_ID         = 4'd5,
               SEND_ENABLE        = 4'd6,
               RECEIVE_ACK_ENABLE = 4'd7,
               STREAM             = 4'd8;

    reg [3:0] state,
              sendState;

    always @ (posedge clk)
        if (reset)
            begin
                state         <= SEND_TIMER_1;
                sendState     <= SEND_RESET;

                dataToSend    <= CODE_RESET;
                send          <= 1;
```

```
        timerStart    <= 1;
        timerTime     <= timerTimeClock;

        decoderEnable <= 0;
    end
else
    begin

        //// Pulse signals ////

        send       <= 0;
        read       <= (state == STREAM) ? 1 : 0;
        timerStart <= 0;

        //// Handle States ////

        case (state)
            SEND_TIMER_1       :    if (timerExpired)
                                        begin
                                            state      <= SEND_TIMER_2;
                                            timerTime  <= timerTimeData;
                                            timerStart <= 1;
                                        end

            SEND_TIMER_2       :    if (timerExpired)
                                        state <= sendState;

            SEND_RESET         :    if (sendFinished)
                                        begin
                                            state      <= RECEIVE_ACK_RESET;
                                            timerTime  <= timerTimePacket;
                                            timerStart <= 1;
                                            read       <= 1;
                                        end

            RECEIVE_ACK_RESET  :    if (timerExpired)
                                        begin
                                            state      <= SEND_TIMER_1;
                                            timerTime  <= timerTimeClock;
                                            timerStart <= 1;
                                            send       <= 1;
                                        end
                                    else if (dataAvailable && dataToReceive == CODE_ACK)
                                        begin
                                            state <= RECEIVE_ACK_BAT;
                                            read  <= 1;
                                        end

            RECEIVE_ACK_BAT    :    if (dataAvailable && dataToReceive == CODE_ACK_BAT)
                                        begin
                                            state <= RECEIVE_ID;
                                            read  <= 1;
                                        end

            RECEIVE_ID         :    if (dataAvailable && dataToReceive == CODE_ID)
                                        begin
                                            state      <= SEND_TIMER_1;
                                            sendState  <= SEND_ENABLE;

                                            dataToSend <= CODE_ENABLE;
                                            send       <= 1;

                                            timerTime  <= timerTimeClock;
                                            timerStart <= 1;
                                        end

            SEND_ENABLE        :    if (sendFinished)
                                        begin
```

```
                                                state      <= RECEIVE_ACK_ENABLE;
                                                timerTime  <= timerTimePacket;
                                                timerStart <= 1;
                                                read       <= 1;
                                            end

                        RECEIVE_ACK_ENABLE :    if (timerExpired)
                                                begin
                                                    state      <= SEND_TIMER_1;
                                                    timerTime  <= timerTimeClock;
                                                    timerStart <= 1;
                                                    send       <= 1;
                                                end
                                            else if (dataAvailable && dataToReceive == CODE_ACK)
                                                state <=  STREAM;

                        STREAM              :   decoderEnable <= 1;

                        default             :   ;
                    endcase
                end

    endmodule

`endif
//newline required
```

11.2.1 TestPS2MouseController

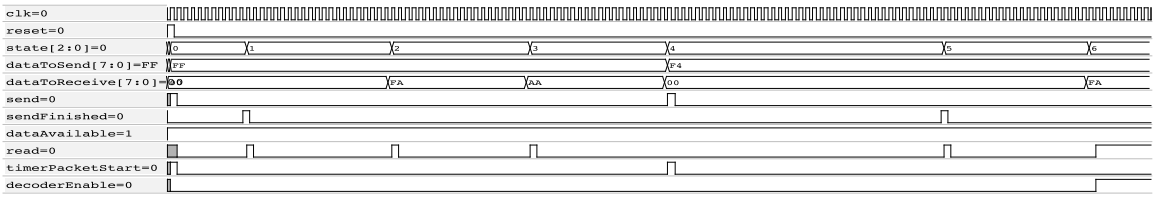

Figure 9: TestPS2MouseController Waveform

11.3 PS2Sender

```
`ifndef INCLUDE_D7B132E1_03D2_4047_985C_4FDE3C6A71FF
    `define INCLUDE_D7B132E1_03D2_4047_985C_4FDE3C6A71FF

    `include "src/synthesis/util/Delay.v"

    module PS2Sender
    (
        input       clk,
        input       reset,

        inout       mouseClock,
        input       mouseClockWentLow,
        inout       mouseData,

        input [7:0] data,
        input       send,

        input       timerExpired,

        output reg  finished
```

```
);

    localparam [1:0] IDLE            = 0,
                     CLOCK_PULL_LOW  = 1,
                     DATA_PULL_LOW   = 2,
                     SENDING         = 3;

    reg [1:0] state;

    reg       mouseClockReg; // use a register to minimize glitches.
    reg       mouseDataReg;

    reg [3:0] dataIndex;

    reg       parity;

    reg       resend;

    always @ (posedge clk)
        if (reset)
            begin
                state           <= IDLE;

                mouseClockReg   <= 1'hz;
                mouseDataReg    <= 1'hz;

                finished        <= 0;
                resend          <= 0;
            end
        else
            begin
                // make signals pulse:
                finished        <= 0;
                resend          <= 0;

                case (state)
                    IDLE:   if (send || resend)
                                begin
                                    state           <= CLOCK_PULL_LOW;
                                    mouseClockReg   <= 0;
                                end

                    CLOCK_PULL_LOW: if (timerExpired)
                                        begin
                                            state          <= DATA_PULL_LOW;
                                            mouseDataReg   <= 0;
                                        end

                    DATA_PULL_LOW:  if (timerExpired)
                                         begin
                                             state         <= SENDING;
                                             dataIndex     <= 0;
                                             parity        <= 1;
                                             mouseClockReg <= 1'hz;
                                         end

                    SENDING:        if (mouseClockWentLow)
                                        begin
                                            dataIndex <= dataIndex+1;

                                            case (dataIndex)
                                                    4'd8: mouseDataReg <= parity;
                                                    4'd9: mouseDataReg <= 1'hz;   // let data line rise to 1 for stop bit.
                                                   4'd10: begin
                                                              state        <= IDLE;

                                                              if (mouseData)
                                                                  resend      <= 1; // should produce an error
                                                              else
```

```
                                                        finished    <= 1;
                                                    end
                                                default: begin
                                                        mouseDataReg <= data[dataIndex];
                                                        parity       <= parity ^ data[dataIndex];
                                                    end
                                            endcase
                                        end

                    endcase
                end

        assign mouseClock = mouseClockReg;
        assign mouseData  = mouseDataReg;

    endmodule

`endif
// newline required
```

11.3.1 TestPS2Sender

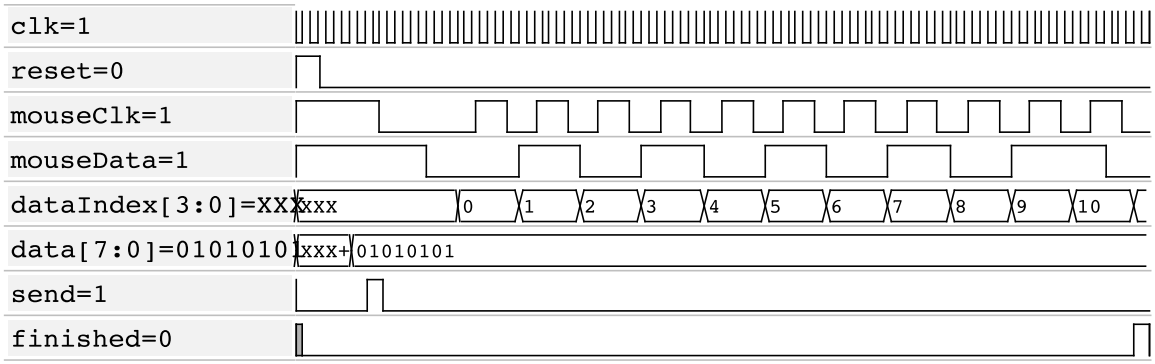

Figure 10: TestPS2Sender Waveform

```
`include "src/synthesis/io/ps2/PS2Sender.v"
`include "src/synthesis/util/LevelToPulseNeg.v"
`include "src/synthesis/util/LevelToPulsePos.v"
`include "src/simulation/Clock.v"

module TestPS2Sender;

    wire clk;
    Clock #(.timeClock(10)) clock(0, clk);

    reg  mouseClockEnable = 0;
    wire clkMouse;

    Clock #(.timeClock(40)) mouseClock(!mouseClockEnable, clkMouse);

    reg mouseDataReg     = 1;

    //iverilog sucks
    wire mouseClkTemp  = mouseClockEnable ? clkMouse     : 1;

    wire (pull1, strong0) mouseClk  = mouseClkTemp;
    wire (pull1, strong0) mouseData = mouseDataReg;

    reg reset        = 1,
```

```
    send         = 0,
    timerExpired = 0;

reg [7:0] data;

wire finished;

wire mouseClockWentLow;
LevelToPulseNeg trackMouseClkNeg
(
    clk,

    reset, 0,

    mouseClk,
    mouseClockWentLow
);

PS2Sender sender
(
    clk,
    reset,

    mouseClk,
    mouseClockWentLow,
    mouseData,

    data, send,

    timerExpired,

    finished
);

initial
    $dumpvars;

integer count = -1;

always @ (posedge clk)
    begin
        timerExpired <= 0;

        if (finished)
            #1 $finish;

        case (clock.cycle)
               2: #3 reset <= 0;
               4: #3 data  <= 8'b01010101;
               5: #3 send  <= 1;
               6: #3 send  <= 0;
               8: #3 timerExpired <= 1;

              10: begin
                      #3 timerExpired     <= 1;
                         mouseClockEnable <= 1;
                  end

              1000: $finish;
        endcase
    end

//// Act Like Mouse ////

always @ (posedge mouseClk)
    case (sender.dataIndex)             // Ack Bit
        10: #3 mouseDataReg <= 0;
        11: #3 mouseDataReg <= 1;
    endcase
```

```
endmodule
```

11.4 PS2Receiver

```
`ifndef INCLUDE_3201A5B1_9EC7_4D3B_A343_62E4D7AEB28E
    `define INCLUDE_3201A5B1_9EC7_4D3B_A343_62E4D7AEB28E

    `include "src/synthesis/util/Delay.v"

    module PS2Receiver
    (
        input             clk,
        input             reset,

        input             mouseClock,
        input             mouseClockWentLow,
        input             mouseDataBit,

        input             read,

        output  reg       byteAvailable,      // byte is available.
        output  reg [7:0] byte
    );

        localparam [1:0] IDLE   = 0,
                         AWAIT  = 1,
                         READ   = 2;

        reg [1:0] state;

        reg [3:0] bitIndex;

        reg [8:0] bits;         // Gathers incoming byte bits and a parity from mouseDataBit.
        reg       bitsInvalid;

        always @ (posedge clk)
            if (reset)
                begin
                    state         <= IDLE;
                    byteAvailable <= 0;
                end
            else
                begin
                    byteAvailable <= 0; // pulse

                    case (state)
                        default :   state <= IDLE;

                        IDLE    :   if (read)
                                        state <= AWAIT;

                        AWAIT   :   if (mouseClockWentLow && !mouseDataBit)           // start bit
                                        begin
                                            state            <= READ;

                                            bitIndex         <= 0;
                                            bitsInvalid      <= 1;
                                        end

                        READ    :   if (mouseClockWentLow)
                                        begin
                                            bitIndex <= bitIndex+1;

                                            if (bitIndex == 9)
                                                begin
```

```
                                        if (mouseDataBit && !bitsInvalid) // stop bit, check byte, bitsInvalid
                                            begin
                                                state         <= IDLE;
                                                byte          <= bits[7:0];
                                                byteAvailable <= 1;
                                            end
                                        else
                                            state <= IDLE;
                                    end
                                else
                                    begin
                                        bits <= {mouseDataBit, bits[8:1]};

                                        // Calculate out own parity; since it finally
                                        // XORed bitsInvalid will be zero if our parity
                                        // matches the parity bit sent by the mouse.
                                        bitsInvalid <= bitsInvalid ^ mouseDataBit;
                                    end
                            end
                    endcase
                end
    endmodule

`endif
//newline required
```

11.4.1 TestPS2Receiver

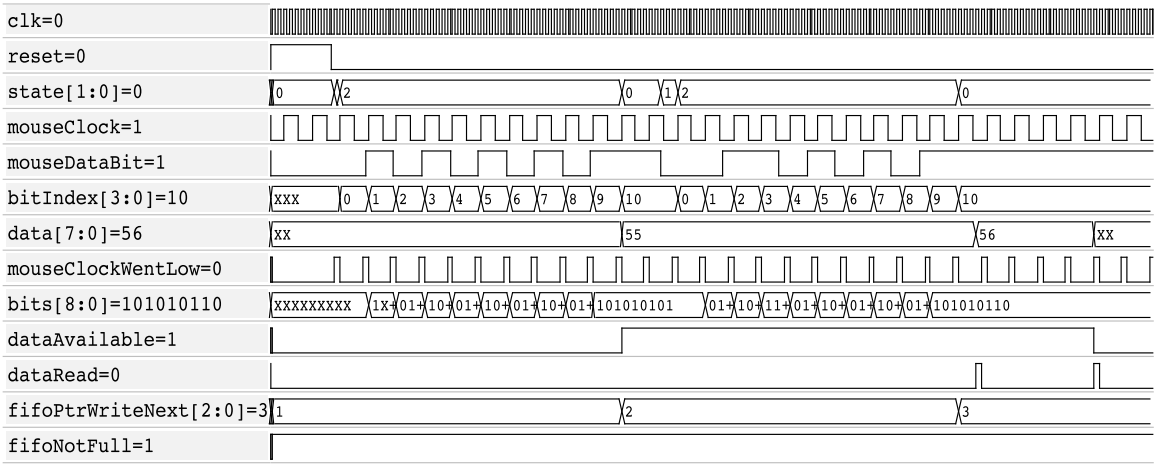

Figure 11: TestPS2Receiver Waveform

11.5 PS2MouseDecoder

```
`ifndef INCLUDE_0A91FCEA_BDA0_424E_8334_1FCEEFA8D39E
    `define INCLUDE_0A91FCEA_BDA0_424E_8334_1FCEEFA8D39E

    module PS2MouseDecoder
    (
        input                   clk,
        input                   reset,

        input                   enable,
```

```
        input             [7:0] byte,
        input                   byteAvailable,

        output reg signed [8:0] deltaX,
        output reg signed [8:0] deltaY,

        output            [2:0] buttons,

        output reg              dataAvailable
    );

        reg    [1:0]    byteIndex;

        reg    [7:0]    status;

        localparam [1:0] BYTE0    = 0,
                         BYTE1    = 1,
                         BYTE2    = 2;

        localparam  BUTTON_LEFT   = 0,
                    BUTTON_RIGHT  = 1,
                    BUTTON_CENTER = 2,
                    SIGN_X        = 4,
                    SIGN_Y        = 5,
                    OVERFLOW_X    = 6,
                    OVERFLOW_Y    = 7;

        localparam  DELTA_MAX     = 8'd255;

        always @ (posedge clk)
            begin
                if (reset)
                    begin
                        status        <= 0;
                        byteIndex     <= 0;
                        deltaX        <= 0;
                        deltaY        <= 0;
                        dataAvailable <= 0;
                    end
                else
                    begin
                        dataAvailable <= 0; // pulse;

                        if (byteAvailable && enable)
                             begin
                                byteIndex <= byteIndex+1;

                                case (byteIndex)
                                    BYTE0:  status <= byte;

                                    BYTE1:  deltaX <= {status[SIGN_X], status[OVERFLOW_X] ? DELTA_MAX : byte};

                                    BYTE2:  begin
                                                deltaY        <= {status[SIGN_Y], status[OVERFLOW_Y] ? DELTA_MAX : byte};
                                                dataAvailable <= 1;
                                                byteIndex     <= 0;
                                            end

                                endcase
                            end
                    end
            end

        assign buttons = {status[BUTTON_LEFT], status[BUTTON_CENTER], status[BUTTON_RIGHT]};

    endmodule

`endif
//newline required
```

11.5.1 TestPS2MouseDecoder

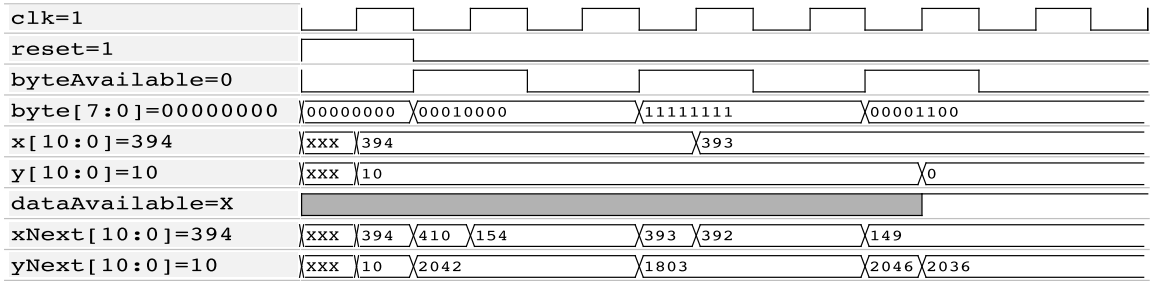

Figure 12: TestPS2MouseDecoder Waveform

12 LabKit

12.1 LabKitPS2Mouse

```
//////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Template Toplevel Module
//
// For Labkit Revision 004
//
//
// Created: October 31, 2004, from revision 003 file
// Author: Nathan Ickes
//
//////////////////////////////////////////////////////////////////////////////
//
// CHANGES FOR BOARD REVISION 004
//
// 1) Added signals for logic analyzer pods 2-4.
// 2) Expanded "tv_in_ycrcb" to 20 bits.
// 3) Renamed "tv_out_data" to "tv_out_i2c_data" and "tv_out_sclk" to
//    "tv_out_i2c_clock".
// 4) Reversed disp_data_in and disp_data_out signals, so that "out" is an
//    output of the FPGA, and "in" is an input.
//
// CHANGES FOR BOARD REVISION 003
//
// 1) Combined flash chip enables into a single signal, flash_ce_b.
//
// CHANGES FOR BOARD REVISION 002
//
// 1) Added SRAM clock feedback path input and output
// 2) Renamed "mousedata" to "mouse_data"
// 3) Renamed some ZBT memory signals. Parity bits are now incorporated into
//    the data bus, and the byte write enables have been combined into the
//    4-bit ram#_bwe_b bus.
// 4) Removed the "systemace_clock" net, since the SystemACE clock is now
//    hardwired on the PCB to the oscillator.
//
//////////////////////////////////////////////////////////////////////////////
//
// Complete change history (including bug fixes)
//
// 2006-Mar-08: Corrected default assignments to "vga_out_red", "vga_out_green"
//              and "vga_out_blue". (Was 10'h0, now 8'h0.)
```

```
//
// 2005-Sep-09: Added missing default assignments to "ac97_sdata_out",
//              "disp_data_out", "analyzer[2-3]_clock" and
//              "analyzer[2-3]_data".
//
// 2005-Jan-23: Reduced flash address bus to 24 bits, to match 128Mb devices
//              actually populated on the boards. (The boards support up to
//              256Mb devices, with 25 address lines.)
//
// 2004-Oct-31: Adapted to new revision 004 board.
//
// 2004-May-01: Changed "disp_data_in" to be an output, and gave it a default
//              value. (Previous versions of this file declared this port to
//              be an input.)
//
// 2004-Apr-29: Reduced SRAM address busses to 19 bits, to match 18Mb devices
//              actually populated on the boards. (The boards support up to
//              72Mb devices, with 21 address lines.)
//
// 2004-Apr-29: Change history started
//
////////////////////////////////////////////////////////////////////////////////

`include "src/synthesis/io/ps2/PS2Mouse.v"
`include "src/synthesis/math/geometry/Rectangle.v"
`include "src/synthesis/math/geometry/rectangle_drawer.v"
`include "src/synthesis/display/VGAController.v"
`include "src/synthesis/util/Delay.v"
`include "src/synthesis/util/Debouncer.v"

module LabKitPS2Mouse (beep, audio_reset_b, ac97_sdata_out, ac97_sdata_in, ac97_synch,
           ac97_bit_clock,

           vga_out_red, vga_out_green, vga_out_blue, vga_out_sync_b,
           vga_out_blank_b, vga_out_pixel_clock, vga_out_hsync,
           vga_out_vsync,

           tv_out_ycrcb, tv_out_reset_b, tv_out_clock, tv_out_i2c_clock,
           tv_out_i2c_data, tv_out_pal_ntsc, tv_out_hsync_b,
           tv_out_vsync_b, tv_out_blank_b, tv_out_subcar_reset,

           tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1,
           tv_in_line_clock2, tv_in_aef, tv_in_hff, tv_in_aff,
           tv_in_i2c_clock, tv_in_i2c_data, tv_in_fifo_read,
           tv_in_fifo_clock, tv_in_iso, tv_in_reset_b, tv_in_clock,

           ram0_data, ram0_address, ram0_adv_ld, ram0_clk, ram0_cen_b,
           ram0_ce_b, ram0_oe_b, ram0_we_b, ram0_bwe_b,

           ram1_data, ram1_address, ram1_adv_ld, ram1_clk, ram1_cen_b,
           ram1_ce_b, ram1_oe_b, ram1_we_b, ram1_bwe_b,

           clock_feedback_out, clock_feedback_in,

           flash_data, flash_address, flash_ce_b, flash_oe_b, flash_we_b,
           flash_reset_b, flash_sts, flash_byte_b,

           rs232_txd, rs232_rxd, rs232_rts, rs232_cts,

           mouse_clock, mouse_data, keyboard_clock, keyboard_data,

           clock_27mhz, clock1, clock2,

           disp_blank, disp_data_out, disp_clock, disp_rs, disp_ce_b,
           disp_reset_b, disp_data_in,

           button0, button1, button2, button3, button_enter, button_right,
           button_left, button_down, button_up,
```

```
       switch,

       led,

       user1, user2, user3, user4,

       daughtercard,

       systemace_data, systemace_address, systemace_ce_b,
       systemace_we_b, systemace_oe_b, systemace_irq, systemace_mpbrdy,

       analyzer1_data, analyzer1_clock,
       analyzer2_data, analyzer2_clock,
       analyzer3_data, analyzer3_clock,
       analyzer4_data, analyzer4_clock);

output beep, audio_reset_b, ac97_synch, ac97_sdata_out;
input  ac97_bit_clock, ac97_sdata_in;

output [7:0] vga_out_red, vga_out_green, vga_out_blue;
output vga_out_sync_b, vga_out_blank_b, vga_out_pixel_clock,
  vga_out_hsync, vga_out_vsync;

output [9:0] tv_out_ycrcb;
output tv_out_reset_b, tv_out_clock, tv_out_i2c_clock, tv_out_i2c_data,
  tv_out_pal_ntsc, tv_out_hsync_b, tv_out_vsync_b, tv_out_blank_b,
  tv_out_subcar_reset;

input  [19:0] tv_in_ycrcb;
input  tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2, tv_in_aef,
  tv_in_hff, tv_in_aff;
output tv_in_i2c_clock, tv_in_fifo_read, tv_in_fifo_clock, tv_in_iso,
  tv_in_reset_b, tv_in_clock;
inout  tv_in_i2c_data;

inout  [35:0] ram0_data;
output [18:0] ram0_address;
output ram0_adv_ld, ram0_clk, ram0_cen_b, ram0_ce_b, ram0_oe_b, ram0_we_b;
output [3:0] ram0_bwe_b;

inout  [35:0] ram1_data;
output [18:0] ram1_address;
output ram1_adv_ld, ram1_clk, ram1_cen_b, ram1_ce_b, ram1_oe_b, ram1_we_b;
output [3:0] ram1_bwe_b;

input  clock_feedback_in;
output clock_feedback_out;

inout  [15:0] flash_data;
output [23:0] flash_address;
output flash_ce_b, flash_oe_b, flash_we_b, flash_reset_b, flash_byte_b;
input  flash_sts;

output rs232_txd, rs232_rts;
input  rs232_rxd, rs232_cts;

inout  mouse_clock, mouse_data;
input  keyboard_clock, keyboard_data;

input  clock_27mhz, clock1, clock2;

output disp_blank, disp_clock, disp_rs, disp_ce_b, disp_reset_b;
input  disp_data_in;
output  disp_data_out;

input  button0, button1, button2, button3, button_enter, button_right,
  button_left, button_down, button_up;
input  [7:0] switch;
output [7:0] led;
```

```
inout [31:0] user1, user2, user3, user4;

inout [43:0] daughtercard;

inout  [15:0] systemace_data;
output [6:0]  systemace_address;
output systemace_ce_b, systemace_we_b, systemace_oe_b;
input  systemace_irq, systemace_mpbrdy;

output [15:0] analyzer1_data, analyzer2_data, analyzer3_data,
     analyzer4_data;
output analyzer1_clock, analyzer2_clock, analyzer3_clock, analyzer4_clock;

////////////////////////////////////////////////////////////////////////
//
// I/O Assignments
//
////////////////////////////////////////////////////////////////////////

// Audio Input and Output
assign beep= 1'b0;
assign audio_reset_b = 1'b0;
assign ac97_synch = 1'b0;
assign ac97_sdata_out = 1'b0;
// ac97_sdata_in is an input

// VGA Output
//assign vga_out_red = 8'h0;
//assign vga_out_green = 8'h0;
//assign vga_out_blue = 8'h0;
//assign vga_out_sync_b = 1'b1;
//assign vga_out_blank_b = 1'b1;
//assign vga_out_pixel_clock = 1'b0;
//assign vga_out_hsync = 1'b0;
//assign vga_out_vsync = 1'b0;

// Video Output
assign tv_out_ycrcb = 10'h0;
assign tv_out_reset_b = 1'b0;
assign tv_out_clock = 1'b0;
assign tv_out_i2c_clock = 1'b0;
assign tv_out_i2c_data = 1'b0;
assign tv_out_pal_ntsc = 1'b0;
assign tv_out_hsync_b = 1'b1;
assign tv_out_vsync_b = 1'b1;
assign tv_out_blank_b = 1'b1;
assign tv_out_subcar_reset = 1'b0;

// Video Input
assign tv_in_i2c_clock = 1'b0;
assign tv_in_fifo_read = 1'b0;
assign tv_in_fifo_clock = 1'b0;
assign tv_in_iso = 1'b0;
assign tv_in_reset_b = 1'b0;
assign tv_in_clock = 1'b0;
assign tv_in_i2c_data = 1'bZ;
// tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2,
// tv_in_aef, tv_in_hff, and tv_in_aff are inputs

// SRAMs
assign ram0_data = 36'hZ;
assign ram0_address = 19'h0;
assign ram0_adv_ld = 1'b0;
assign ram0_clk = 1'b0;
assign ram0_cen_b = 1'b1;
assign ram0_ce_b = 1'b1;
assign ram0_oe_b = 1'b1;
assign ram0_we_b = 1'b1;
```

```
assign ram0_bwe_b = 4'hF;
assign ram1_data = 36'hZ;
assign ram1_address = 19'h0;
assign ram1_adv_ld = 1'b0;
assign ram1_clk = 1'b0;
assign ram1_cen_b = 1'b1;
assign ram1_ce_b = 1'b1;
assign ram1_oe_b = 1'b1;
assign ram1_we_b = 1'b1;
assign ram1_bwe_b = 4'hF;
assign clock_feedback_out = 1'b0;
// clock_feedback_in is an input

// Flash ROM
assign flash_data = 16'hZ;
assign flash_address = 24'h0;
assign flash_ce_b = 1'b1;
assign flash_oe_b = 1'b1;
assign flash_we_b = 1'b1;
assign flash_reset_b = 1'b0;
assign flash_byte_b = 1'b1;
// flash_sts is an input

// RS-232 Interface
assign rs232_txd = 1'b1;
assign rs232_rts = 1'b1;
// rs232_rxd and rs232_cts are inputs

// PS/2 Ports
// mouse_clock, mouse_data, keyboard_clock, and keyboard_data are inputs

// LED Displays
assign disp_blank = 1'b1;
assign disp_clock = 1'b0;
assign disp_rs = 1'b0;
assign disp_ce_b = 1'b1;
assign disp_reset_b = 1'b0;
assign disp_data_out = 1'b0;
// disp_data_in is an input

// Buttons, Switches, and Individual LEDs
//assign led = 8'hFF;
// button0, button1, button2, button3, button_enter, button_right,
// button_left, button_down, button_up, and switches are inputs

// User I/Os
assign user1 = 32'hZ;
assign user2 = 32'hZ;
assign user3 = 32'hZ;
assign user4 = 32'hZ;

// Daughtercard Connectors
assign daughtercard = 44'hZ;

// SystemACE Microprocessor Port
assign systemace_data = 16'hZ;
assign systemace_address = 7'h0;
assign systemace_ce_b = 1'b1;
assign systemace_we_b = 1'b1;
assign systemace_oe_b = 1'b1;
// systemace_irq and systemace_mpbrdy are inputs

// Logic Analyzer
assign analyzer1_data = 16'h0;
assign analyzer1_clock = 1'b1;
assign analyzer2_data = 16'h0;
assign analyzer2_clock = 1'b1;
assign analyzer3_data = 16'h0;
assign analyzer3_clock = 1'b1;
```

```
    assign analyzer4_data = 16'h0;
    assign analyzer4_clock = 1'b1;


    ////////////////////////////// END DEFAULTS //////////////////////////

    ////////////////////
//// Pixel Clock ////
////////////////////

wire clk;
DCM pixel_clock_dcm (.CLKIN(clock_27mhz), .CLKFX(clk));
// synthesis attribute CLKFX_MULTIPLY of pixel_clock_dcm is 31
// synthesis attribute CLKFX_DIVIDE of pixel_clock_dcm is 21
// synthesis attribute CLKIN_PERIOD of pixel_clock_dcm is 37
// synthesis attribute CLK_FEEDBACK of pixel_clock_dcm is NONE

// Send the inverted clock to the DAC,
// to allow for enough setup time;
// Otherwise clock skew could mess everything up.
assign vga_out_pixel_clock = !clk;

///////////////
//// Reset ////
///////////////

reg        resetPowerOn = 1;
reg [26:0] timer = 0;

always @ (posedge clock_27mhz)
   if (timer == 53999999)
       resetPowerOn <= 0;
   else
       timer <= timer+1;

wire resetUser;
Debouncer #(.delay(400000)) debounceReset(clk, resetPowerOn, ~button_enter, resetUser);

wire reset = resetUser | resetPowerOn;

////////////////////////
//// VGA Controller ////
////////////////////////

wire syncH, syncV;

wire [10:0] pixelX;
wire [9:0]  pixelY;

VGAController
    #(.hActive(800), .hPorchFront(40), .hSync(128), .hPorchBack(88),  .hBits(11),
      .vActive(600), .vPorchFront(1),  .vSync(4),   .vPorchBack(23),  .vBits(10))
vga
    (
clk,
reset,

vga_out_blank_b,
syncH,
syncV,

pixelX, pixelY
);

Delay #(.delay(2)) syncHDelay(clk, syncH, vga_out_hsync);
Delay #(.delay(2)) syncVDelay(clk, syncV, vga_out_vsync);

//// Unused signals ////
    assign vga_out_sync_b = 1;      // This is historical and unnecessary.
```

```
    assign vga_out_green  = 8'hz;
    assign vga_out_blue   = 8'hz;

    ////////////////////////
//// PS/2 Interface ////
////////////////////////

/*reg next;
always @ (posedge clk)
   if (reset)
       next <= 0;
   else
       begin
           next <= 0; // pulse it.

           if (dataAvailable && !next)
                   next <= 1;
           end
    */

wire bufferFull,
     dataAvailable,
     next;

    Debouncer debounceNext(clk, reset, ~button0, next);

wire [10:0] mouseX;
wire [9:0] mouseY;
wire [2:0] buttons;

PS2Mouse mouse
    (
        clk,
        reset,

        mouse_clock,
        mouse_data,

        next,

        mouseX,
        mouseY,
        buttons,

        bufferFull,
        dataAvailable
    );

    /////////////////
//// Display ////
/////////////////

wire isInside;
Rectangle #(.bitsX(11), .bitsY(10))
cursor
(
   reset,

   mouseX, mouseY,
   10, 10,
   pixelX, pixelY,

   isInside
);

    assign vga_out_red = {8{isInside}};

assign led = {~bufferFull, ~dataAvailable, 3'b111, ~buttons};
```

```
endmodule
```

12.2 LabKitZBT

```
//////////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Template Toplevel Module
//
// For Labkit Revision 004
//
//
// Created: October 31, 2004, from revision 003 file
// Author: Nathan Ickes
//
//////////////////////////////////////////////////////////////////////////////////
//
// CHANGES FOR BOARD REVISION 004
//
// 1) Added signals for logic analyzer pods 2-4.
// 2) Expanded "tv_in_ycrcb" to 20 bits.
// 3) Renamed "tv_out_data" to "tv_out_i2c_data" and "tv_out_sclk" to
//    "tv_out_i2c_clock".
// 4) Reversed disp_data_in and disp_data_out signals, so that "out" is an
//    output of the FPGA, and "in" is an input.
//
// CHANGES FOR BOARD REVISION 003
//
// 1) Combined flash chip enables into a single signal, flash_ce_b.
//
// CHANGES FOR BOARD REVISION 002
//
// 1) Added SRAM clock feedback path input and output
// 2) Renamed "mousedata" to "mouse_data"
// 3) Renamed some ZBT memory signals. Parity bits are now incorporated into
//    the data bus, and the byte write enables have been combined into the
//    4-bit ram#_bwe_b bus.
// 4) Removed the "systemace_clock" net, since the SystemACE clock is now
//    hardwired on the PCB to the oscillator.
//
//////////////////////////////////////////////////////////////////////////////////
//
// Complete change history (including bug fixes)
//
// 2006-Mar-08: Corrected default assignments to "vga_out_red", "vga_out_green"
//              and "vga_out_blue". (Was 10'h0, now 8'h0.)
//
// 2005-Sep-09: Added missing default assignments to "ac97_sdata_out",
//              "disp_data_out", "analyzer[2-3]_clock" and
//              "analyzer[2-3]_data".
//
// 2005-Jan-23: Reduced flash address bus to 24 bits, to match 128Mb devices
//              actually populated on the boards. (The boards support up to
//              256Mb devices, with 25 address lines.)
//
// 2004-Oct-31: Adapted to new revision 004 board.
//
// 2004-May-01: Changed "disp_data_in" to be an output, and gave it a default
//              value. (Previous versions of this file declared this port to
//              be an input.)
//
// 2004-Apr-29: Reduced SRAM address busses to 19 bits, to match 18Mb devices
//              actually populated on the boards. (The boards support up to
//              72Mb devices, with 21 address lines.)
//
// 2004-Apr-29: Change history started
//
```

```
//////////////////////////////////////////////////////////////////////////////

`include "src/synthesis/memory/ZBT.v"
`include "src/synthesis/math/geometry/Rectangle.v"
`include "src/synthesis/display/VGAController.v"
`include "src/synthesis/util/Delay.v"
`include "src/synthesis/util/Debouncer.v"

module LabKitZBT (beep, audio_reset_b, ac97_sdata_out, ac97_sdata_in, ac97_synch,
           ac97_bit_clock,

           vga_out_red, vga_out_green, vga_out_blue, vga_out_sync_b,
           vga_out_blank_b, vga_out_pixel_clock, vga_out_hsync,
           vga_out_vsync,

           tv_out_ycrcb, tv_out_reset_b, tv_out_clock, tv_out_i2c_clock,
           tv_out_i2c_data, tv_out_pal_ntsc, tv_out_hsync_b,
           tv_out_vsync_b, tv_out_blank_b, tv_out_subcar_reset,

           tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1,
           tv_in_line_clock2, tv_in_aef, tv_in_hff, tv_in_aff,
           tv_in_i2c_clock, tv_in_i2c_data, tv_in_fifo_read,
           tv_in_fifo_clock, tv_in_iso, tv_in_reset_b, tv_in_clock,

           ram0_data, ram0_address, ram0_adv_ld, ram0_clk, ram0_cen_b,
           ram0_ce_b, ram0_oe_b, ram0_we_b, ram0_bwe_b,

           ram1_data, ram1_address, ram1_adv_ld, ram1_clk, ram1_cen_b,
           ram1_ce_b, ram1_oe_b, ram1_we_b, ram1_bwe_b,

           clock_feedback_out, clock_feedback_in,

           flash_data, flash_address, flash_ce_b, flash_oe_b, flash_we_b,
           flash_reset_b, flash_sts, flash_byte_b,

           rs232_txd, rs232_rxd, rs232_rts, rs232_cts,

           mouse_clock, mouse_data, keyboard_clock, keyboard_data,

           clock_27mhz, clock1, clock2,

           disp_blank, disp_data_out, disp_clock, disp_rs, disp_ce_b,
           disp_reset_b, disp_data_in,

           button0, button1, button2, button3, button_enter, button_right,
           button_left, button_down, button_up,

           switch,

           led,

           user1, user2, user3, user4,

           daughtercard,

           systemace_data, systemace_address, systemace_ce_b,
           systemace_we_b, systemace_oe_b, systemace_irq, systemace_mpbrdy,

           analyzer1_data, analyzer1_clock,
           analyzer2_data, analyzer2_clock,
           analyzer3_data, analyzer3_clock,
           analyzer4_data, analyzer4_clock);

    output beep, audio_reset_b, ac97_synch, ac97_sdata_out;
    input  ac97_bit_clock, ac97_sdata_in;

    output [7:0] vga_out_red, vga_out_green, vga_out_blue;
    output vga_out_sync_b, vga_out_blank_b, vga_out_pixel_clock,
      vga_out_hsync, vga_out_vsync;
```

```
output [9:0] tv_out_ycrcb;
output tv_out_reset_b, tv_out_clock, tv_out_i2c_clock, tv_out_i2c_data,
  tv_out_pal_ntsc, tv_out_hsync_b, tv_out_vsync_b, tv_out_blank_b,
  tv_out_subcar_reset;

input  [19:0] tv_in_ycrcb;
input  tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2, tv_in_aef,
  tv_in_hff, tv_in_aff;
output tv_in_i2c_clock, tv_in_fifo_read, tv_in_fifo_clock, tv_in_iso,
  tv_in_reset_b, tv_in_clock;
inout  tv_in_i2c_data;

inout  [35:0] ram0_data;
output [18:0] ram0_address;
output ram0_adv_ld, ram0_clk, ram0_cen_b, ram0_ce_b, ram0_oe_b, ram0_we_b;
output [3:0] ram0_bwe_b;

inout  [35:0] ram1_data;
output [18:0] ram1_address;
output ram1_adv_ld, ram1_clk, ram1_cen_b, ram1_ce_b, ram1_oe_b, ram1_we_b;
output [3:0] ram1_bwe_b;

input  clock_feedback_in;
output clock_feedback_out;

inout  [15:0] flash_data;
output [23:0] flash_address;
output flash_ce_b, flash_oe_b, flash_we_b, flash_reset_b, flash_byte_b;
input  flash_sts;

output rs232_txd, rs232_rts;
input  rs232_rxd, rs232_cts;

inout  mouse_clock, mouse_data;
input  keyboard_clock, keyboard_data;

input  clock_27mhz, clock1, clock2;

output disp_blank, disp_clock, disp_rs, disp_ce_b, disp_reset_b;
input  disp_data_in;
output  disp_data_out;

input  button0, button1, button2, button3, button_enter, button_right,
  button_left, button_down, button_up;
input  [7:0] switch;
output [7:0] led;

inout [31:0] user1, user2, user3, user4;

inout [43:0] daughtercard;

inout  [15:0] systemace_data;
output [6:0]  systemace_address;
output systemace_ce_b, systemace_we_b, systemace_oe_b;
input  systemace_irq, systemace_mpbrdy;

output [15:0] analyzer1_data, analyzer2_data, analyzer3_data,
     analyzer4_data;
output analyzer1_clock, analyzer2_clock, analyzer3_clock, analyzer4_clock;

////////////////////////////////////////////////////////////////////////
//
// I/O Assignments
//
////////////////////////////////////////////////////////////////////////

// Audio Input and Output
assign beep= 1'b0;
```

```
    assign audio_reset_b = 1'b0;
    assign ac97_synch = 1'b0;
    assign ac97_sdata_out = 1'b0;
    // ac97_sdata_in is an input

    // VGA Output
    //assign vga_out_red = 8'h0;
    //assign vga_out_green = 8'h0;
    //assign vga_out_blue = 8'h0;
    //assign vga_out_sync_b = 1'b1;
    //assign vga_out_blank_b = 1'b1;
    //assign vga_out_pixel_clock = 1'b0;
    //assign vga_out_hsync = 1'b0;
    //assign vga_out_vsync = 1'b0;

    // Video Output
    assign tv_out_ycrcb = 10'h0;
    assign tv_out_reset_b = 1'b0;
    assign tv_out_clock = 1'b0;
    assign tv_out_i2c_clock = 1'b0;
    assign tv_out_i2c_data = 1'b0;
    assign tv_out_pal_ntsc = 1'b0;
    assign tv_out_hsync_b = 1'b1;
    assign tv_out_vsync_b = 1'b1;
    assign tv_out_blank_b = 1'b1;
    assign tv_out_subcar_reset = 1'b0;

    // Video Input
    assign tv_in_i2c_clock = 1'b0;
    assign tv_in_fifo_read = 1'b0;
    assign tv_in_fifo_clock = 1'b0;
    assign tv_in_iso = 1'b0;
    assign tv_in_reset_b = 1'b0;
    assign tv_in_clock = 1'b0;
    assign tv_in_i2c_data = 1'bZ;
    // tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2,
    // tv_in_aef, tv_in_hff, and tv_in_aff are inputs

    // SRAMs
//    assign ram0_data = 36'hZ;
//    assign ram0_address = 19'h0;
//    assign ram0_adv_ld = 1'b0;
//    assign ram0_clk = 1'b0;
//    assign ram0_cen_b = 1'b1;
//    assign ram0_ce_b = 1'b1;
//    assign ram0_oe_b = 1'b1;
//    assign ram0_we_b = 1'b1;
//    assign ram0_bwe_b = 4'hF;
    assign ram1_data = 36'hZ;
    assign ram1_address = 19'h0;
    assign ram1_adv_ld = 1'b0;
    assign ram1_clk = 1'b0;
    assign ram1_cen_b = 1'b1;
    assign ram1_ce_b = 1'b1;
    assign ram1_oe_b = 1'b1;
    assign ram1_we_b = 1'b1;
    assign ram1_bwe_b = 4'hF;
    assign clock_feedback_out = 1'b0;
    // clock_feedback_in is an input

    // Flash ROM
    assign flash_data = 16'hZ;
    assign flash_address = 24'h0;
    assign flash_ce_b = 1'b1;
    assign flash_oe_b = 1'b1;
    assign flash_we_b = 1'b1;
    assign flash_reset_b = 1'b0;
    assign flash_byte_b = 1'b1;
    // flash_sts is an input
```

```
    // RS-232 Interface
    assign rs232_txd = 1'b1;
    assign rs232_rts = 1'b1;
    // rs232_rxd and rs232_cts are inputs

    // PS/2 Ports
    // mouse_clock, mouse_data, keyboard_clock, and keyboard_data are inputs

    // LED Displays
    assign disp_blank = 1'b1;
    assign disp_clock = 1'b0;
    assign disp_rs = 1'b0;
    assign disp_ce_b = 1'b1;
    assign disp_reset_b = 1'b0;
    assign disp_data_out = 1'b0;
    // disp_data_in is an input

    // Buttons, Switches, and Individual LEDs
    assign led = 8'hFF;
    // button0, button1, button2, button3, button_enter, button_right,
    // button_left, button_down, button_up, and switches are inputs

    // User I/Os
    assign user1 = 32'hZ;
    assign user2 = 32'hZ;
    assign user3 = 32'hZ;
    assign user4 = 32'hZ;

    // Daughtercard Connectors
    assign daughtercard = 44'hZ;

    // SystemACE Microprocessor Port
    assign systemace_data = 16'hZ;
    assign systemace_address = 7'h0;
    assign systemace_ce_b = 1'b1;
    assign systemace_we_b = 1'b1;
    assign systemace_oe_b = 1'b1;
    // systemace_irq and systemace_mpbrdy are inputs

    // Logic Analyzer
    //assign analyzer1_data = 16'h0;
    assign analyzer1_clock = 1'b1;
    assign analyzer2_data = 16'h0;
    assign analyzer2_clock = 1'b1;
    assign analyzer3_data = 16'h0;
    assign analyzer3_clock = 1'b1;
    assign analyzer4_data = 16'h0;
    assign analyzer4_clock = 1'b1;


    ////////////////////////////// END DEFAULTS //////////////////////////

    ////////////////////
//// Pixel Clock ////
////////////////////

wire clk;
DCM pixel_clock_dcm (.CLKIN(clock_27mhz), .CLKFX(clk));
// synthesis attribute CLKFX_MULTIPLY of pixel_clock_dcm is 31
// synthesis attribute CLKFX_DIVIDE of pixel_clock_dcm is 21
// synthesis attribute CLKIN_PERIOD of pixel_clock_dcm is 37
// synthesis attribute CLK_FEEDBACK of pixel_clock_dcm is NONE

// Send the inverted clock to the DAC,
// to allow for enough setup time;
// Otherwise clock skew could mess everything up.
assign vga_out_pixel_clock = !clk;
```

```
///////////////
//// Reset ////
///////////////

reg        resetPowerOn = 1;
reg [26:0] timer = 0;

always @ (posedge clock_27mhz)
   if (timer == 53999999)
       resetPowerOn <= 0;
   else
       timer <= timer+1;

wire resetUser;
Debouncer #(.delay(400000)) debounceReset(clk, resetPowerOn, ~button_enter, resetUser);

wire reset = resetUser | resetPowerOn;

////////////////////////
//// VGA Controller ////
////////////////////////

wire syncH, syncV;

wire [10:0] pixelX;
wire [9:0]  pixelY;

VGAController
    #(.hActive(800), .hPorchFront(40), .hSync(128), .hPorchBack(88),  .hBits(11),
      .vActive(600), .vPorchFront(1),  .vSync(4),   .vPorchBack(23),  .vBits(10))
vga
    (
clk,
reset,

vga_out_blank_b,
syncH,
syncV,

pixelX, pixelY
);

Delay #(.delay(2)) syncHDelay(clk, syncH, vga_out_hsync);
Delay #(.delay(2)) syncVDelay(clk, syncV, vga_out_vsync);

//// Unused signals ////
    assign vga_out_sync_b = 1;      // This is historical and unnecessary.
    assign vga_out_green  = 8'hz;
    assign vga_out_blue   = 8'hz;


    /////////////
//// ZBT ////
/////////////

    reg  [18:0] address;

    reg         read;

    reg  [35:0] dataToWrite;
    wire [35:0] dataToRead;

    wire clkZBT;

ZBT ramZBT
(
   clk,
   reset,
```

```
   read,

       address,
   dataToWrite,
   dataToRead,

       ram0_address,
       ram0_data,
   ram0_ce_b,
   ram0_oe_b,
   ram0_we_b,
   ram0_bwe_b,
   ram0_cen_b,
   ram0_adv_ld,
   clkZBT
    );

    assign ram0_clk = ~clkZBT; // This is easier than having to use a phase lock loop.

    //reg [1:0] counter;

    always @ (posedge clk)
        if (reset)
            begin
                read    <= 0;
                address <= 0;
                //counter <= 0;
            end
        else
            begin
                if (!read && (pixelX < 800) && (pixelY < 600))
                    begin
                        address <= address+1;
                        dataToWrite <= {15'b0, pixelX, pixelY};
                    end

                if (pixelX == (1055-3) && pixelY == 627)                  // Be wary of pipelining.
                    begin
                        if (read)
                            begin
                                //counter <= counter+1;
                                //if (counter == 3)
                                    address <= address+1;
                            end
                        else
                            read <= 1;
                    end

                if (address == 479999)
                    address <= 0;
            end

    /////////////////
//// Display ////
/////////////////

wire isInside;
Rectangle #(.bitsX(11), .bitsY(10))
cursor
(
   reset,

   dataToRead[20:10], dataToRead[9:0],
   10, 10,
   pixelX, pixelY,

   isInside
);
```

```
    assign vga_out_red = {8{isInside}};

    ////////////////////////
//// Logic Analyzer ////
////////////////////////

    assign analyzer1_data = {7'b0, reset, clk, read, dataToWrite[1:0], ram0_address[1:0], dataToRead[1:0]};

endmodule
```

12.3 LabKitScheduler

```
`include "src/synthesis/control/InteractionScheduler.v"
`include "src/synthesis/util/Debouncer.v"
`include "src/synthesis/util/Enabler.v"

//////////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Template Toplevel Module
//
// For Labkit Revision 004
//
//
// Created: October 31, 2004, from revision 003 file
// Author: Nathan Ickes
//
//////////////////////////////////////////////////////////////////////////////////
//
// CHANGES FOR BOARD REVISION 004
//
// 1) Added signals for logic analyzer pods 2-4.
// 2) Expanded "tv_in_ycrcb" to 20 bits.
// 3) Renamed "tv_out_data" to "tv_out_i2c_data" and "tv_out_sclk" to
//    "tv_out_i2c_clock".
// 4) Reversed disp_data_in and disp_data_out signals, so that "out" is an
//    output of the FPGA, and "in" is an input.
//
// CHANGES FOR BOARD REVISION 003
//
// 1) Combined flash chip enables into a single signal, flash_ce_b.
//
// CHANGES FOR BOARD REVISION 002
//
// 1) Added SRAM clock feedback path input and output
// 2) Renamed "mousedata" to "mouse_data"
// 3) Renamed some ZBT memory signals. Parity bits are now incorporated into
//    the data bus, and the byte write enables have been combined into the
//    4-bit ram#_bwe_b bus.
// 4) Removed the "systemace_clock" net, since the SystemACE clock is now
//    hardwired on the PCB to the oscillator.
//
//////////////////////////////////////////////////////////////////////////////////
//
// Complete change history (including bug fixes)
//
// 2006-Mar-08: Corrected default assignments to "vga_out_red", "vga_out_green"
//              and "vga_out_blue". (Was 10'h0, now 8'h0.)
//
// 2005-Sep-09: Added missing default assignments to "ac97_sdata_out",
//              "disp_data_out", "analyzer[2-3]_clock" and
//              "analyzer[2-3]_data".
//
// 2005-Jan-23: Reduced flash address bus to 24 bits, to match 128Mb devices
//              actually populated on the boards. (The boards support up to
//              256Mb devices, with 25 address lines.)
//
```

```
// 2004-Oct-31: Adapted to new revision 004 board.
//
// 2004-May-01: Changed "disp_data_in" to be an output, and gave it a default
//              value. (Previous versions of this file declared this port to
//              be an input.)
//
// 2004-Apr-29: Reduced SRAM address busses to 19 bits, to match 18Mb devices
//              actually populated on the boards. (The boards support up to
//              72Mb devices, with 21 address lines.)
//
// 2004-Apr-29: Change history started
//
////////////////////////////////////////////////////////////////////////////////

module LabKitInteractionScheduler (beep, audio_reset_b, ac97_sdata_out, ac97_sdata_in, ac97_synch,
       ac97_bit_clock,

       vga_out_red, vga_out_green, vga_out_blue, vga_out_sync_b,
       vga_out_blank_b, vga_out_pixel_clock, vga_out_hsync,
       vga_out_vsync,

       tv_out_ycrcb, tv_out_reset_b, tv_out_clock, tv_out_i2c_clock,
       tv_out_i2c_data, tv_out_pal_ntsc, tv_out_hsync_b,
       tv_out_vsync_b, tv_out_blank_b, tv_out_subcar_reset,

       tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1,
       tv_in_line_clock2, tv_in_aef, tv_in_hff, tv_in_aff,
       tv_in_i2c_clock, tv_in_i2c_data, tv_in_fifo_read,
       tv_in_fifo_clock, tv_in_iso, tv_in_reset_b, tv_in_clock,

       ram0_data, ram0_address, ram0_adv_ld, ram0_clk, ram0_cen_b,
       ram0_ce_b, ram0_oe_b, ram0_we_b, ram0_bwe_b,

       ram1_data, ram1_address, ram1_adv_ld, ram1_clk, ram1_cen_b,
       ram1_ce_b, ram1_oe_b, ram1_we_b, ram1_bwe_b,

       clock_feedback_out, clock_feedback_in,

       flash_data, flash_address, flash_ce_b, flash_oe_b, flash_we_b,
       flash_reset_b, flash_sts, flash_byte_b,

       rs232_txd, rs232_rxd, rs232_rts, rs232_cts,

       mouse_clock, mouse_data, keyboard_clock, keyboard_data,

       clock_27mhz, clock1, clock2,

       disp_blank, disp_data_out, disp_clock, disp_rs, disp_ce_b,
       disp_reset_b, disp_data_in,

       button0, button1, button2, button3, button_enter, button_right,
       button_left, button_down, button_up,

       switch,

       led,

       user1, user2, user3, user4,

       daughtercard,

       systemace_data, systemace_address, systemace_ce_b,
       systemace_we_b, systemace_oe_b, systemace_irq, systemace_mpbrdy,

       analyzer1_data, analyzer1_clock,
         analyzer2_data, analyzer2_clock,
         analyzer3_data, analyzer3_clock,
         analyzer4_data, analyzer4_clock);
```

```
  output beep, audio_reset_b, ac97_synch, ac97_sdata_out;
  input  ac97_bit_clock, ac97_sdata_in;

  output [7:0] vga_out_red, vga_out_green, vga_out_blue;
  output vga_out_sync_b, vga_out_blank_b, vga_out_pixel_clock,
 vga_out_hsync, vga_out_vsync;

  output [9:0] tv_out_ycrcb;
  output tv_out_reset_b, tv_out_clock, tv_out_i2c_clock, tv_out_i2c_data,
 tv_out_pal_ntsc, tv_out_hsync_b, tv_out_vsync_b, tv_out_blank_b,
 tv_out_subcar_reset;

  input  [19:0] tv_in_ycrcb;
  input  tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2, tv_in_aef,
 tv_in_hff, tv_in_aff;
  output tv_in_i2c_clock, tv_in_fifo_read, tv_in_fifo_clock, tv_in_iso,
 tv_in_reset_b, tv_in_clock;
  inout  tv_in_i2c_data;

  inout  [35:0] ram0_data;
  output [18:0] ram0_address;
  output ram0_adv_ld, ram0_clk, ram0_cen_b, ram0_ce_b, ram0_oe_b, ram0_we_b;
  output [3:0] ram0_bwe_b;

  inout  [35:0] ram1_data;
  output [18:0] ram1_address;
  output ram1_adv_ld, ram1_clk, ram1_cen_b, ram1_ce_b, ram1_oe_b, ram1_we_b;
  output [3:0] ram1_bwe_b;

  input  clock_feedback_in;
  output clock_feedback_out;

  inout  [15:0] flash_data;
  output [23:0] flash_address;
  output flash_ce_b, flash_oe_b, flash_we_b, flash_reset_b, flash_byte_b;
  input  flash_sts;

  output rs232_txd, rs232_rts;
  input  rs232_rxd, rs232_cts;

  input  mouse_clock, mouse_data, keyboard_clock, keyboard_data;

  input  clock_27mhz, clock1, clock2;

  output disp_blank, disp_clock, disp_rs, disp_ce_b, disp_reset_b;
  input  disp_data_in;
  output  disp_data_out;

  input  button0, button1, button2, button3, button_enter, button_right,
 button_left, button_down, button_up;
  input  [7:0] switch;
  output [7:0] led;

  inout [31:0] user1, user2, user3, user4;

  inout [43:0] daughtercard;

  inout  [15:0] systemace_data;
  output [6:0]  systemace_address;
  output systemace_ce_b, systemace_we_b, systemace_oe_b;
  input  systemace_irq, systemace_mpbrdy;

  output [15:0] analyzer1_data, analyzer2_data, analyzer3_data,
analyzer4_data;
  output analyzer1_clock, analyzer2_clock, analyzer3_clock, analyzer4_clock;

  ////////////////////////////////////////////////////////////////////////
  //
  // I/O Assignments
```

```
//
//////////////////////////////////////////////////////////////////////////////

// Audio Input and Output
assign beep= 1'b0;
assign audio_reset_b = 1'b0;
assign ac97_synch = 1'b0;
assign ac97_sdata_out = 1'b0;

// VGA Output
assign vga_out_red = 8'h0;
assign vga_out_green = 8'h0;
assign vga_out_blue = 8'h0;
assign vga_out_sync_b = 1'b1;
assign vga_out_blank_b = 1'b1;
assign vga_out_pixel_clock = 1'b0;
assign vga_out_hsync = 1'b0;
assign vga_out_vsync = 1'b0;

// Video Output
assign tv_out_ycrcb = 10'h0;
assign tv_out_reset_b = 1'b0;
assign tv_out_clock = 1'b0;
assign tv_out_i2c_clock = 1'b0;
assign tv_out_i2c_data = 1'b0;
assign tv_out_pal_ntsc = 1'b0;
assign tv_out_hsync_b = 1'b1;
assign tv_out_vsync_b = 1'b1;
assign tv_out_blank_b = 1'b1;
assign tv_out_subcar_reset = 1'b0;

// Video Input
assign tv_in_i2c_clock = 1'b0;
assign tv_in_fifo_read = 1'b0;
assign tv_in_fifo_clock = 1'b0;
assign tv_in_iso = 1'b0;
assign tv_in_reset_b = 1'b0;
assign tv_in_clock = 1'b0;
assign tv_in_i2c_data = 1'bZ;

// SRAMs
assign ram0_data = 36'hZ;
assign ram0_address = 19'h0;
assign ram0_adv_ld = 1'b0;
assign ram0_clk = 1'b0;
assign ram0_cen_b = 1'b1;
assign ram0_ce_b = 1'b1;
assign ram0_oe_b = 1'b1;
assign ram0_we_b = 1'b1;
assign ram0_bwe_b = 4'hF;
assign ram1_data = 36'hZ;
assign ram1_address = 19'h0;
assign ram1_adv_ld = 1'b0;
assign ram1_clk = 1'b0;
assign ram1_cen_b = 1'b1;
assign ram1_ce_b = 1'b1;
assign ram1_oe_b = 1'b1;
assign ram1_we_b = 1'b1;
assign ram1_bwe_b = 4'hF;
assign clock_feedback_out = 1'b0;

// Flash ROM
assign flash_data = 16'hZ;
assign flash_address = 24'h0;
assign flash_ce_b = 1'b1;
assign flash_oe_b = 1'b1;
assign flash_we_b = 1'b1;
assign flash_reset_b = 1'b0;
assign flash_byte_b = 1'b1;
```

```
   // RS-232 Interface
   assign rs232_txd = 1'b1;
   assign rs232_rts = 1'b1;

   // LED Displays
//   assign disp_blank = 1'b1;
//   assign disp_clock = 1'b0;
//   assign disp_rs = 1'b0;
//   assign disp_ce_b = 1'b1;
//   assign disp_reset_b = 1'b0;
//   assign disp_data_out = 1'b0;

   // Buttons, Switches, and Individual LEDs
   assign led = 8'hFF;

   // User I/Os
   assign user1 = 32'hZ;
   assign user2 = 32'hZ;
   assign user3 = 32'hZ;
   assign user4 = 32'hZ;

   // Daughtercard Connectors
   assign daughtercard = 44'hZ;

   // SystemACE Microprocessor Port
   assign systemace_data = 16'hZ;
   assign systemace_address = 7'h0;
   assign systemace_ce_b = 1'b1;
   assign systemace_we_b = 1'b1;
   assign systemace_oe_b = 1'b1;

   // Logic Analyzer
   assign analyzer1_data = 16'h0;
   assign analyzer1_clock = 1'b1;
   assign analyzer2_data = 16'h0;
   assign analyzer2_clock = 1'b1;
   assign analyzer3_data = 16'h0;
   assign analyzer3_clock = 1'b1;
   assign analyzer4_data = 16'h0;
   assign analyzer4_clock = 1'b1;

   ///////////////
//// Reset ////
///////////////

reg        resetPowerOn = 1;
reg [26:0] timer = 0;

always @ (posedge clock_27mhz)
   if (timer == 53999999)
       resetPowerOn <= 0;
   else
       timer <= timer+1;

wire resetUser;
Debouncer debounceReset(clk, resetPowerOn, ~button_enter, resetUser);

wire reset = resetUser | resetPowerOn;

    wire [3:0] primary, secondary;

    wire finished;

    wire enable;

    Enabler count (clock_27mhz, reset, enable);

    InteractionScheduler
```

```
    #(.objects(10), .objectsBits(4), .accumulatorStages(3))
    scheduler
    (
        enable,
        reset,

        primary,
        secondary,

        finished
    );

    wire [79:0] primDots, secDots;

    dots dots1 (enable, primary, primDots);
    dots dots2 (enable, secondary, secDots);

    display disp
    (
        reset, clock_27mhz,

        disp_blank,
        disp_clock,
        disp_rs,
        disp_ce_b,
disp_reset_b,
        disp_data_out,

        {primDots, {480{1'b0}}, secDots}
     );

endmodule

//////////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Alphanumeric Display Interface
//
//
// Created: November 5, 2003
// Author: Nathan Ickes
//
//////////////////////////////////////////////////////////////////////////////////
//
// Change history
//
// 2007-02-09: Fixed register-select race condition. (Thanks to Chris
//             Buenrostro for finding this bug and David Wentzloff for
//             implementing the fix.)
// 2005-05-09: Made <dots> input registered, and converted the 640-input MUX
//             to a 640-bit shift register.
//
//////////////////////////////////////////////////////////////////////////////////

module display (reset, clock_27mhz, disp_blank, disp_clock, disp_rs, disp_ce_b,
disp_reset_b, disp_data_out, dots);

   input reset; // Active high
   input clock_27mhz;
   output disp_blank, disp_clock, disp_data_out, disp_rs, disp_ce_b,
  disp_reset_b;
   input [639:0] dots;

   reg disp_data_out, disp_rs, disp_ce_b, disp_reset_b;

   //
   // Display Clock
   //
   // Generate a 500kHz clock for driving the displays.
   //
```

```
   reg [4:0] count;
   reg [7:0] reset_count;
   reg clock;
   wire dreset;

   always @(posedge clock_27mhz)
     begin
if (reset)
  begin
     count = 0;
     clock = 0;
  end
else if (count == 26)
  begin
     clock = ~clock;
     count = 5'h00;
  end
else
  count = count+1;
     end

   always @(posedge clock_27mhz)
     if (reset)
       reset_count <= 100;
     else
       reset_count <= (reset_count==0) ? 0 : reset_count-1;

   assign dreset = (reset_count != 0);

   assign disp_clock = ~clock;

   //
   // Display State Machine
   //

   reg [7:0] state;
   reg [9:0] dot_index;
   reg [31:0] control;
   reg [639:0] ldots;

   assign disp_blank = 1'b0; // low = not blanked

   always @(posedge clock)
     if (dreset)
       begin
  state <= 0;
  dot_index <= 0;
  control <= 32'h7F7F7F7F;
       end
     else
       casex (state)
 8'h00:
   begin
      // Reset displays
      disp_data_out <= 1'b0;
      disp_rs <= 1'b0; // 0 = dot register
      disp_ce_b <= 1'b1;
      disp_reset_b <= 1'b0;
      dot_index <= 0;
      state <= state+1;
   end

 8'h01:
   begin
      // End reset
      disp_reset_b <= 1'b1;
      state <= state+1;
   end
```

```
 8'h02:
   begin
      // Initialize dot register
      disp_ce_b <= 1'b0;
      disp_data_out <= 1'b0; // dot_index[0];
      if (dot_index == 639)
state <= state+1;
      else
dot_index <= dot_index+1;
   end

 8'h03:
   begin
      // Latch dot data
      disp_rs <= 1'b1; // Select the control register
      disp_ce_b <= 1'b1;
      dot_index <= 31;
      state <= state+1;
   end

 8'h04:
   begin
      // Setup the control register
      disp_ce_b <= 1'b0;
      disp_data_out <= control[31];
      control <= {control[30:0], 1'b0};
      if (dot_index == 0)
state <= state+1;
      else
dot_index <= dot_index-1;
   end

 8'h05:
   begin
      // Latch the control register data
      disp_rs <= 1'b0; // Select the dot register
      disp_ce_b <= 1'b1;
      dot_index <= 639;
      ldots <= dots;
      state <= state+1;
   end

 8'h06:
   begin
      // Load the user's dot data into the dot register
      disp_ce_b <= 1'b0;
      disp_data_out <= ldots[639];
      ldots <= ldots<<1;
      if (dot_index == 0)
state <= 5;
      else
dot_index <= dot_index-1;
   end
       endcase

endmodule

//////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Number to bitmap decoder
//
//
// Author: Yun Wu, Nathan Ickes
// Date: March 8, 2006
//
// This module converts a 4-bit input to a 80-dot (2 digit) bitmap representing
// the numbers ' 0' through '15'.
//
```

```
//////////////////////////////////////////////////////////////////////////////////

module dots(clk, num, dots);
   input clk;
   input [3:0] num;
   output [79:0] dots;

   reg [79:0] dots;
   always @ (posedge clk)
     case (num)
       4'd15: dots <= {40'b00000000_01000010_01111111_01000000_00000000, // '15'
       40'b00100111_01000101_01000101_01000101_00111001};
       4'd14: dots <= {40'b00000000_01000010_01111111_01000000_00000000, // '14'
       40'b00011000_00010100_00010010_01111111_00010000};
       4'd13: dots <= {40'b00000000_01000010_01111111_01000000_00000000, // '13'
       40'b00100010_01000001_01001001_01001001_00110110};
       4'd12: dots <= {40'b00000000_01000010_01111111_01000000_00000000, // '12'
       40'b01100010_01010001_01001001_01001001_01000110};
       4'd11: dots <= {40'b00000000_01000010_01111111_01000000_00000000, // '11'
       40'b00000000_01000010_01111111_01000000_00000000};
       4'd10: dots <= {40'b00000000_01000010_01111111_01000000_00000000, // '10'
       40'b00111110_01010001_01001001_01000101_00111110};
       4'd09: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 9'
       40'b00000110_01001001_01001001_00101001_00011110};
       4'd08: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 8'
       40'b00110110_01001001_01001001_01001001_00110110};
       4'd07: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 7'
       40'b00000001_01110001_00001001_00000101_00000011};
       4'd06: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 6'
       40'b00111100_01001010_01001001_01001001_00110000};
       4'd05: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 5'
       40'b00100111_01000101_01000101_01000101_00111001};
       4'd04: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 4'
       40'b00011000_00010100_00010010_01111111_00010000};
       4'd03: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 3'
       40'b00100010_01000001_01001001_01001001_00110110};
       4'd02: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 2'
       40'b01100010_01010001_01001001_01001001_01000110};
       4'd01: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 1'
       40'b00000000_01000010_01111111_01000000_00000000};
       4'd00: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 0'
       40'b00111110_01010001_01001001_01000101_00111110};
       // No default case, becase every case is already accounted for.
     endcase

endmodule
```

12.4 LabKitDrawerCircle

12.4.1 TestLabKitDrawerCircle

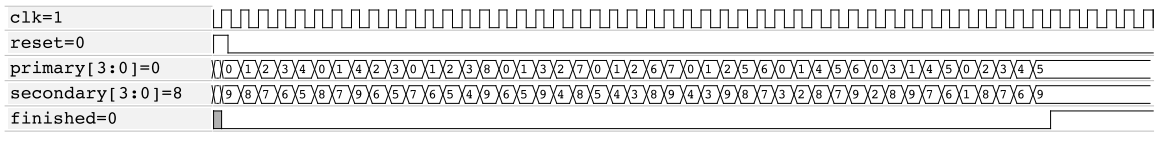

Figure 13:

```
`include "src/synthesis/io/ps2/PS2Mouse.v"
`include "src/synthesis/math/geometry/Rectangle.v"
```

```
`include "src/synthesis/display/VGAController.v"
`include "src/synthesis/util/Delay.v"
`include "src/synthesis/util/Debouncer.v"
`include "src/synthesis/memory/ZBT.v"
`include "src/synthesis/display/DrawerCircle.v"

`include "src/simulation/Clock.v"

module TestLabKitDrawerCircle;

    wire clock_27mhz;
    Clock #(.timeClock(3)) fake_clock_27mhz(0, clock_27mhz);

    wire clk;
    Clock #(.timeClock(2)) dcm(0, clk);

    tri  [35:0] ram0_data;
    wire [18:0] ram0_address;
    wire [3:0]  ram0_bwe_b;
    wire        ram0_adv_ld,
                ram0_clk,
                ram0_cen_b,
                ram0_ce_b,
                ram0_oe_b,
                ram0_we_b;

    wire        vga_out_blank_b,
                vga_out_sync_b,
                vga_out_hsync,
                vga_out_vsync,
                vga_out_red,
                vga_out_green,
                vga_out_blue;

    reg        button_enter  = 1;

    ////////////////////////////////////////////////////

    ///////////////
    //// Reset ////
    ///////////////

    reg        resetPowerOn = 1;
    reg [26:0] timer        = 0;

    always @ (posedge clock_27mhz)
       //if (timer == 53999999)
       if (timer == 10)
           resetPowerOn <= 0;
       else
           timer <= timer+1;

    wire resetUser;
    Debouncer #(.delay(1/*400000*/)) debounceReset(clk, resetPowerOn, ~button_enter, resetUser);

    wire reset = resetUser | resetPowerOn;


    /////////////
    //// ZBT ////
    /////////////

    wire [18:0] memoryAddress;

    wire        memoryRead;

    wire [35:0] dataToWrite;
    wire [35:0] dataToRead;
```

```
    wire        clkZBT;

    ZBT memoryZBT
    (
       clk,
       reset,

       memoryRead,

       memoryAddress,
       dataToWrite,
       dataToRead,

       ram0_address,
       ram0_data,
       ram0_ce_b,
       ram0_oe_b,
       ram0_we_b,
       ram0_bwe_b,
       ram0_cen_b,
       ram0_adv_ld,
       clkZBT
    );

    assign ram0_clk = ~clkZBT; // This is easier than having to use a phase lock loop.

    ////////////////////////
    //// VGA Controller ////
    ////////////////////////

//  localparam WIDTH  = 200, H_PORCH_FRONT = 40, H_SYNC = 128, H_PORCH_BACK = 88, H_BITS = 11,
//             HEIGHT = 200, V_PORCH_FRONT =  1, V_SYNC =   4, V_PORCH_BACK = 23, V_BITS = 10,

    localparam WIDTH  = 10, H_PORCH_FRONT =  1,  H_SYNC = 1, H_PORCH_BACK = 1,  H_BITS = 11,
               HEIGHT = 10, V_PORCH_FRONT =  1,  V_SYNC = 1, V_PORCH_BACK = 1,  V_BITS = 10,

               H_PIXELS = WIDTH  + H_PORCH_FRONT + H_SYNC + H_PORCH_BACK,
               V_PIXELS = HEIGHT + V_PORCH_FRONT + V_SYNC + V_PORCH_BACK;

    wire syncH, syncV;

    wire [10:0] pixelX;
    wire [ 9:0] pixelY;

    VGAController
    #(.hActive(WIDTH),  .hPorchFront(H_PORCH_FRONT), .hSync(H_SYNC), .hPorchBack(H_PORCH_BACK),  .hBits(H_BITS),
      .vActive(HEIGHT), .vPorchFront(V_PORCH_FRONT), .vSync(V_SYNC), .vPorchBack(V_PORCH_BACK),  .vBits(V_BITS))
    vga
    (
        clk,
        reset,

        vga_out_blank_b,
        syncH,
        syncV,

        pixelX, pixelY
    );

    Delay #(.delay(2)) syncHDelay(clk, syncH, vga_out_hsync);
    Delay #(.delay(2)) syncVDelay(clk, syncV, vga_out_vsync);

    //// Unused signals ////
    assign vga_out_sync_b = 1;      // This is historical and unnecessary.

    assign vga_out_red   = dataToRead[23:16];
    assign vga_out_green = dataToRead[15: 8];
    assign vga_out_blue  = dataToRead[ 7: 0];
```

```
//////////////////////
//// PS/2 Interface ////
//////////////////////

/*wire bufferFull,
     dataAvailable,
     next;

Debouncer debounceNext(clk, reset, ~button0, next);

wire [10:0] mouseX;
wire [9:0]  mouseY;
wire [2:0]  buttons;

PS2Mouse mouse
(
    clk,
    reset,

    mouse_clock,
    mouse_data,

    next,

    mouseX,
    mouseY,
    buttons,

    bufferFull,
    dataAvailable
);

/////////////////
//// Display ////
/////////////////

wire isInside;
Rectangle #(.bitsX(11), .bitsY(10))
cursor
(
   reset,

   mouseX, mouseY,
   10, 10,
   pixelX, pixelY,

   isInside
);

assign vga_out_red = {8{isInside}};

assign led = {~bufferFull, ~dataAvailable, 3'b111, ~buttons};*/

////////////////
//// Circle ////
////////////////

reg signed  'INT  centerX,
                  centerY;

reg         'INT  radius;

reg               start;

wire [18:0]       circleAddress;

wire              writeDisable;
wire              finished;
```

```
DrawerCircle #(.width(WIDTH), .height(HEIGHT)) drawer
(
    clk,
    reset,

    start,

    centerX,
    centerY,

    radius,

    circleAddress,
    writeDisable,

    finished
);

///////////////
//// Logic ////
///////////////

localparam RADIUS     = 5;

localparam CLEARING   = 2'd0,
           DRAWING    = 2'd1,
           DISPLAYING = 2'd2;

reg [1:0]  state;

reg [18:0] logicAddress;

always @ (posedge clk)
    begin
        start        <= 0;
        logicAddress <= logicAddress+1;

        if (reset)
            begin
                state        <= CLEARING;

                logicAddress <= 0;

                centerX      <= RADIUS;
                centerY      <= RADIUS;

                radius       <= RADIUS;
            end
        else
            begin
                case (state)
                    CLEARING   :    if (pixelX == WIDTH-2 && pixelY == HEIGHT-1)
                                        begin
                                            state <= DRAWING;
                                            start <= 1;
                                        end

                    DRAWING    :    if (pixelX == (H_PIXELS-3) && pixelY == V_PIXELS-1)                 // Be wary of pipel
                                        begin
                                            state        <= DISPLAYING;
                                            logicAddress <= 0;
                                        end

                    DISPLAYING :    if (pixelX == WIDTH-2 && pixelY == HEIGHT-1)
                                        begin
                                            state <= DRAWING;
                                            start <= 1;
                                        end
                endcase
```

```
                end
        end

    assign memoryAddress = (state==DRAWING)    ? circleAddress : logicAddress,
           dataToWrite   = (state==DRAWING)    ?     36'hFF    :      0      ,
           memoryRead    = (state==DISPLAYING) ?       1       :      0      ;

    ////////////////////////////////////////////////////////////

    initial
        $dumpvars;

    integer finishedCount = 0;

    always @ (posedge clk)
        begin
            if (finished)
                finishedCount <= finishedCount+1;

            if (finishedCount == 3)
                #10 $finish;
        end

endmodule
```

12.5 LabKitCircleDrawerDoubleBuffered

12.5.1 TestLabKitCircleDrawerDoubleBuffered

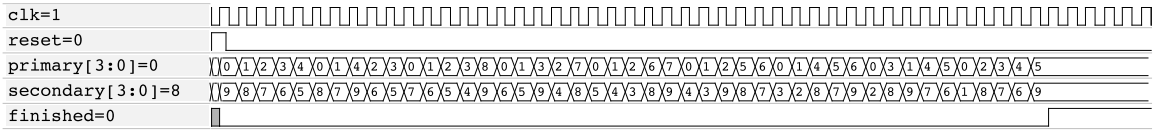

Figure 14:

12.6 LabKitScheduler

```
`include "src/synthesis/control/InteractionScheduler.v"
`include "src/synthesis/util/Debouncer.v"
`include "src/synthesis/util/Enabler.v"

////////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Template Toplevel Module
//
// For Labkit Revision 004
//
//
// Created: October 31, 2004, from revision 003 file
// Author: Nathan Ickes
//
////////////////////////////////////////////////////////////////////////////////
//
// CHANGES FOR BOARD REVISION 004
//
// 1) Added signals for logic analyzer pods 2-4.
// 2) Expanded "tv_in_ycrcb" to 20 bits.
```

```
// 3) Renamed "tv_out_data" to "tv_out_i2c_data" and "tv_out_sclk" to
//    "tv_out_i2c_clock".
// 4) Reversed disp_data_in and disp_data_out signals, so that "out" is an
//    output of the FPGA, and "in" is an input.
//
// CHANGES FOR BOARD REVISION 003
//
// 1) Combined flash chip enables into a single signal, flash_ce_b.
//
// CHANGES FOR BOARD REVISION 002
//
// 1) Added SRAM clock feedback path input and output
// 2) Renamed "mousedata" to "mouse_data"
// 3) Renamed some ZBT memory signals. Parity bits are now incorporated into
//    the data bus, and the byte write enables have been combined into the
//    4-bit ram#_bwe_b bus.
// 4) Removed the "systemace_clock" net, since the SystemACE clock is now
//    hardwired on the PCB to the oscillator.
//
//////////////////////////////////////////////////////////////////////////////
//
// Complete change history (including bug fixes)
//
// 2006-Mar-08: Corrected default assignments to "vga_out_red", "vga_out_green"
//              and "vga_out_blue". (Was 10'h0, now 8'h0.)
//
// 2005-Sep-09: Added missing default assignments to "ac97_sdata_out",
//              "disp_data_out", "analyzer[2-3]_clock" and
//              "analyzer[2-3]_data".
//
// 2005-Jan-23: Reduced flash address bus to 24 bits, to match 128Mb devices
//              actually populated on the boards. (The boards support up to
//              256Mb devices, with 25 address lines.)
//
// 2004-Oct-31: Adapted to new revision 004 board.
//
// 2004-May-01: Changed "disp_data_in" to be an output, and gave it a default
//              value. (Previous versions of this file declared this port to
//              be an input.)
//
// 2004-Apr-29: Reduced SRAM address busses to 19 bits, to match 18Mb devices
//              actually populated on the boards. (The boards support up to
//              72Mb devices, with 21 address lines.)
//
// 2004-Apr-29: Change history started
//
//////////////////////////////////////////////////////////////////////////////

module LabKitInteractionScheduler (beep, audio_reset_b, ac97_sdata_out, ac97_sdata_in, ac97_synch,
       ac97_bit_clock,

       vga_out_red, vga_out_green, vga_out_blue, vga_out_sync_b,
       vga_out_blank_b, vga_out_pixel_clock, vga_out_hsync,
       vga_out_vsync,

       tv_out_ycrcb, tv_out_reset_b, tv_out_clock, tv_out_i2c_clock,
       tv_out_i2c_data, tv_out_pal_ntsc, tv_out_hsync_b,
       tv_out_vsync_b, tv_out_blank_b, tv_out_subcar_reset,

       tv_in_ycrcb, tv_in_data_valid, tv_in_line_clock1,
       tv_in_line_clock2, tv_in_aef, tv_in_hff, tv_in_aff,
       tv_in_i2c_clock, tv_in_i2c_data, tv_in_fifo_read,
       tv_in_fifo_clock, tv_in_iso, tv_in_reset_b, tv_in_clock,

       ram0_data, ram0_address, ram0_adv_ld, ram0_clk, ram0_cen_b,
       ram0_ce_b, ram0_oe_b, ram0_we_b, ram0_bwe_b,

       ram1_data, ram1_address, ram1_adv_ld, ram1_clk, ram1_cen_b,
       ram1_ce_b, ram1_oe_b, ram1_we_b, ram1_bwe_b,
```

```
     clock_feedback_out, clock_feedback_in,

     flash_data, flash_address, flash_ce_b, flash_oe_b, flash_we_b,
     flash_reset_b, flash_sts, flash_byte_b,

     rs232_txd, rs232_rxd, rs232_rts, rs232_cts,

     mouse_clock, mouse_data, keyboard_clock, keyboard_data,

     clock_27mhz, clock1, clock2,

     disp_blank, disp_data_out, disp_clock, disp_rs, disp_ce_b,
     disp_reset_b, disp_data_in,

     button0, button1, button2, button3, button_enter, button_right,
     button_left, button_down, button_up,

     switch,

     led,

     user1, user2, user3, user4,

     daughtercard,

     systemace_data, systemace_address, systemace_ce_b,
     systemace_we_b, systemace_oe_b, systemace_irq, systemace_mpbrdy,

     analyzer1_data, analyzer1_clock,
       analyzer2_data, analyzer2_clock,
       analyzer3_data, analyzer3_clock,
       analyzer4_data, analyzer4_clock);

 output beep, audio_reset_b, ac97_synch, ac97_sdata_out;
 input  ac97_bit_clock, ac97_sdata_in;

 output [7:0] vga_out_red, vga_out_green, vga_out_blue;
 output vga_out_sync_b, vga_out_blank_b, vga_out_pixel_clock,
vga_out_hsync, vga_out_vsync;

 output [9:0] tv_out_ycrcb;
 output tv_out_reset_b, tv_out_clock, tv_out_i2c_clock, tv_out_i2c_data,
tv_out_pal_ntsc, tv_out_hsync_b, tv_out_vsync_b, tv_out_blank_b,
tv_out_subcar_reset;

 input  [19:0] tv_in_ycrcb;
 input  tv_in_data_valid, tv_in_line_clock1, tv_in_line_clock2, tv_in_aef,
tv_in_hff, tv_in_aff;
 output tv_in_i2c_clock, tv_in_fifo_read, tv_in_fifo_clock, tv_in_iso,
tv_in_reset_b, tv_in_clock;
 inout  tv_in_i2c_data;

 inout  [35:0] ram0_data;
 output [18:0] ram0_address;
 output ram0_adv_ld, ram0_clk, ram0_cen_b, ram0_ce_b, ram0_oe_b, ram0_we_b;
 output [3:0] ram0_bwe_b;

 inout  [35:0] ram1_data;
 output [18:0] ram1_address;
 output ram1_adv_ld, ram1_clk, ram1_cen_b, ram1_ce_b, ram1_oe_b, ram1_we_b;
 output [3:0] ram1_bwe_b;

 input  clock_feedback_in;
 output clock_feedback_out;

 inout  [15:0] flash_data;
 output [23:0] flash_address;
 output flash_ce_b, flash_oe_b, flash_we_b, flash_reset_b, flash_byte_b;
```

```
  input  flash_sts;

  output rs232_txd, rs232_rts;
  input  rs232_rxd, rs232_cts;

  input  mouse_clock, mouse_data, keyboard_clock, keyboard_data;

  input  clock_27mhz, clock1, clock2;

  output disp_blank, disp_clock, disp_rs, disp_ce_b, disp_reset_b;
  input  disp_data_in;
  output  disp_data_out;

  input  button0, button1, button2, button3, button_enter, button_right,
 button_left, button_down, button_up;
  input  [7:0] switch;
  output [7:0] led;

  inout [31:0] user1, user2, user3, user4;

  inout [43:0] daughtercard;

  inout  [15:0] systemace_data;
  output [6:0]  systemace_address;
  output systemace_ce_b, systemace_we_b, systemace_oe_b;
  input  systemace_irq, systemace_mpbrdy;

  output [15:0] analyzer1_data, analyzer2_data, analyzer3_data,
analyzer4_data;
  output analyzer1_clock, analyzer2_clock, analyzer3_clock, analyzer4_clock;

  ////////////////////////////////////////////////////////////////////////
  //
  // I/O Assignments
  //
  ////////////////////////////////////////////////////////////////////////

  // Audio Input and Output
  assign beep= 1'b0;
  assign audio_reset_b = 1'b0;
  assign ac97_synch = 1'b0;
  assign ac97_sdata_out = 1'b0;

  // VGA Output
  assign vga_out_red = 8'h0;
  assign vga_out_green = 8'h0;
  assign vga_out_blue = 8'h0;
  assign vga_out_sync_b = 1'b1;
  assign vga_out_blank_b = 1'b1;
  assign vga_out_pixel_clock = 1'b0;
  assign vga_out_hsync = 1'b0;
  assign vga_out_vsync = 1'b0;

  // Video Output
  assign tv_out_ycrcb = 10'h0;
  assign tv_out_reset_b = 1'b0;
  assign tv_out_clock = 1'b0;
  assign tv_out_i2c_clock = 1'b0;
  assign tv_out_i2c_data = 1'b0;
  assign tv_out_pal_ntsc = 1'b0;
  assign tv_out_hsync_b = 1'b1;
  assign tv_out_vsync_b = 1'b1;
  assign tv_out_blank_b = 1'b1;
  assign tv_out_subcar_reset = 1'b0;

  // Video Input
  assign tv_in_i2c_clock = 1'b0;
  assign tv_in_fifo_read = 1'b0;
  assign tv_in_fifo_clock = 1'b0;
```

```
   assign tv_in_iso = 1'b0;
   assign tv_in_reset_b = 1'b0;
   assign tv_in_clock = 1'b0;
   assign tv_in_i2c_data = 1'bZ;

   // SRAMs
   assign ram0_data = 36'hZ;
   assign ram0_address = 19'h0;
   assign ram0_adv_ld = 1'b0;
   assign ram0_clk = 1'b0;
   assign ram0_cen_b = 1'b1;
   assign ram0_ce_b = 1'b1;
   assign ram0_oe_b = 1'b1;
   assign ram0_we_b = 1'b1;
   assign ram0_bwe_b = 4'hF;
   assign ram1_data = 36'hZ;
   assign ram1_address = 19'h0;
   assign ram1_adv_ld = 1'b0;
   assign ram1_clk = 1'b0;
   assign ram1_cen_b = 1'b1;
   assign ram1_ce_b = 1'b1;
   assign ram1_oe_b = 1'b1;
   assign ram1_we_b = 1'b1;
   assign ram1_bwe_b = 4'hF;
   assign clock_feedback_out = 1'b0;

   // Flash ROM
   assign flash_data = 16'hZ;
   assign flash_address = 24'h0;
   assign flash_ce_b = 1'b1;
   assign flash_oe_b = 1'b1;
   assign flash_we_b = 1'b1;
   assign flash_reset_b = 1'b0;
   assign flash_byte_b = 1'b1;

   // RS-232 Interface
   assign rs232_txd = 1'b1;
   assign rs232_rts = 1'b1;

   // LED Displays
//   assign disp_blank = 1'b1;
//   assign disp_clock = 1'b0;
//   assign disp_rs = 1'b0;
//   assign disp_ce_b = 1'b1;
//   assign disp_reset_b = 1'b0;
//   assign disp_data_out = 1'b0;

   // Buttons, Switches, and Individual LEDs
   assign led = 8'hFF;

   // User I/Os
   assign user1 = 32'hZ;
   assign user2 = 32'hZ;
   assign user3 = 32'hZ;
   assign user4 = 32'hZ;

   // Daughtercard Connectors
   assign daughtercard = 44'hZ;

   // SystemACE Microprocessor Port
   assign systemace_data = 16'hZ;
   assign systemace_address = 7'h0;
   assign systemace_ce_b = 1'b1;
   assign systemace_we_b = 1'b1;
   assign systemace_oe_b = 1'b1;

   // Logic Analyzer
   assign analyzer1_data = 16'h0;
   assign analyzer1_clock = 1'b1;
```

```
   assign analyzer2_data = 16'h0;
   assign analyzer2_clock = 1'b1;
   assign analyzer3_data = 16'h0;
   assign analyzer3_clock = 1'b1;
   assign analyzer4_data = 16'h0;
   assign analyzer4_clock = 1'b1;

   ///////////////
//// Reset ////
///////////////

reg        resetPowerOn = 1;
reg [26:0] timer = 0;

always @ (posedge clock_27mhz)
   if (timer == 53999999)
       resetPowerOn <= 0;
   else
       timer <= timer+1;

wire resetUser;
Debouncer debounceReset(clk, resetPowerOn, ~button_enter, resetUser);

wire reset = resetUser | resetPowerOn;

    wire [3:0] primary, secondary;

    wire finished;

    wire enable;

    Enabler count (clock_27mhz, reset, enable);

    InteractionScheduler
    #(.objects(10), .objectsBits(4), .accumulatorStages(3))
    scheduler
    (
        enable,
        reset,

        primary,
        secondary,

        finished
    );

    wire [79:0] primDots, secDots;

    dots dots1 (enable, primary, primDots);
    dots dots2 (enable, secondary, secDots);

    display disp
    (
        reset, clock_27mhz,

        disp_blank,
        disp_clock,
        disp_rs,
        disp_ce_b,
disp_reset_b,
        disp_data_out,

        {primDots, {480{1'b0}}, secDots}
     );

endmodule

//////////////////////////////////////////////////////////////////////////////////
//
```

```
// 6.111 FPGA Labkit -- Alphanumeric Display Interface
//
//
// Created: November 5, 2003
// Author: Nathan Ickes
//
////////////////////////////////////////////////////////////////////////////////
//
// Change history
//
// 2007-02-09: Fixed register-select race condition. (Thanks to Chris
//             Buenrostro for finding this bug and David Wentzloff for
//             implementing the fix.)
// 2005-05-09: Made <dots> input registered, and converted the 640-input MUX
//             to a 640-bit shift register.
//
////////////////////////////////////////////////////////////////////////////////

module display (reset, clock_27mhz, disp_blank, disp_clock, disp_rs, disp_ce_b,
disp_reset_b, disp_data_out, dots);

   input reset; // Active high
   input clock_27mhz;
   output disp_blank, disp_clock, disp_data_out, disp_rs, disp_ce_b,
  disp_reset_b;
   input [639:0] dots;

   reg disp_data_out, disp_rs, disp_ce_b, disp_reset_b;

   //
   // Display Clock
   //
   // Generate a 500kHz clock for driving the displays.
   //

   reg [4:0] count;
   reg [7:0] reset_count;
   reg clock;
   wire dreset;

   always @(posedge clock_27mhz)
     begin
if (reset)
  begin
     count = 0;
     clock = 0;
  end
else if (count == 26)
  begin
     clock = ~clock;
     count = 5'h00;
  end
else
  count = count+1;
     end

   always @(posedge clock_27mhz)
     if (reset)
       reset_count <= 100;
     else
       reset_count <= (reset_count==0) ? 0 : reset_count-1;

   assign dreset = (reset_count != 0);

   assign disp_clock = ~clock;

   //
   // Display State Machine
   //
```

```
   reg [7:0] state;
   reg [9:0] dot_index;
   reg [31:0] control;
   reg [639:0] ldots;

   assign disp_blank = 1'b0; // low = not blanked

   always @(posedge clock)
     if (dreset)
       begin
  state <= 0;
  dot_index <= 0;
  control <= 32'h7F7F7F7F;
       end
     else
       casex (state)
 8'h00:
   begin
      // Reset displays
      disp_data_out <= 1'b0;
      disp_rs <= 1'b0; // 0 = dot register
      disp_ce_b <= 1'b1;
      disp_reset_b <= 1'b0;
      dot_index <= 0;
      state <= state+1;
   end

 8'h01:
   begin
      // End reset
      disp_reset_b <= 1'b1;
      state <= state+1;
   end

 8'h02:
   begin
      // Initialize dot register
      disp_ce_b <= 1'b0;
      disp_data_out <= 1'b0; // dot_index[0];
      if (dot_index == 639)
state <= state+1;
      else
dot_index <= dot_index+1;
   end

 8'h03:
   begin
      // Latch dot data
      disp_rs <= 1'b1; // Select the control register
      disp_ce_b <= 1'b1;
      dot_index <= 31;
      state <= state+1;
   end

 8'h04:
   begin
      // Setup the control register
      disp_ce_b <= 1'b0;
      disp_data_out <= control[31];
      control <= {control[30:0], 1'b0};
      if (dot_index == 0)
state <= state+1;
      else
dot_index <= dot_index-1;
   end

 8'h05:
   begin
```

```
      // Latch the control register data
      disp_rs <= 1'b0; // Select the dot register
      disp_ce_b <= 1'b1;
      dot_index <= 639;
      ldots <= dots;
      state <= state+1;
   end

 8'h06:
   begin
      // Load the user's dot data into the dot register
      disp_ce_b <= 1'b0;
      disp_data_out <= ldots[639];
      ldots <= ldots<<1;
      if (dot_index == 0)
state <= 5;
      else
dot_index <= dot_index-1;
   end
       endcase

endmodule

//////////////////////////////////////////////////////////////////////////////////
//
// 6.111 FPGA Labkit -- Number to bitmap decoder
//
//
// Author: Yun Wu, Nathan Ickes
// Date: March 8, 2006
//
// This module converts a 4-bit input to a 80-dot (2 digit) bitmap representing
// the numbers ' 0' through '15'.
//
//////////////////////////////////////////////////////////////////////////////////

module dots(clk, num, dots);
   input clk;
   input [3:0] num;
   output [79:0] dots;

   reg [79:0] dots;
   always @ (posedge clk)
     case (num)
       4'd15: dots <= {40'b00000000_01000010_01111111_01000000_00000000, // '15'
       40'b00100111_01000101_01000101_01000101_00111001};
       4'd14: dots <= {40'b00000000_01000010_01111111_01000000_00000000, // '14'
       40'b00011000_00010100_00010010_01111111_00010000};
       4'd13: dots <= {40'b00000000_01000010_01111111_01000000_00000000, // '13'
       40'b00100010_01000001_01001001_01001001_00110110};
       4'd12: dots <= {40'b00000000_01000010_01111111_01000000_00000000, // '12'
       40'b01100010_01010001_01001001_01001001_01000110};
       4'd11: dots <= {40'b00000000_01000010_01111111_01000000_00000000, // '11'
       40'b00000000_01000010_01111111_01000000_00000000};
       4'd10: dots <= {40'b00000000_01000010_01111111_01000000_00000000, // '10'
       40'b00111110_01010001_01001001_01000101_00111110};
       4'd09: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 9'
       40'b00000110_01001001_01001001_00101001_00011110};
       4'd08: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 8'
       40'b00110110_01001001_01001001_01001001_00110110};
       4'd07: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 7'
       40'b00000001_01110001_00001001_00000101_00000011};
       4'd06: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 6'
       40'b00111100_01001010_01001001_01001001_00110000};
       4'd05: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 5'
       40'b00100111_01000101_01000101_01000101_00111001};
       4'd04: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 4'
       40'b00011000_00010100_00010010_01111111_00010000};
       4'd03: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 3'
```

```
      40'b00100010_01000001_01001001_01001001_00110110};
      4'd02: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 2'
      40'b01100010_01010001_01001001_01001001_01000110};
      4'd01: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 1'
      40'b00000000_01000010_01111111_01000000_00000000};
      4'd00: dots <= {40'b00000000_00000000_00000000_00000000_00000000, // ' 0'
      40'b00111110_01010001_01001001_01000101_00111110};
      // No default case, becase every case is already accounted for.
    endcase

endmodule
```

13 Utilities

13.1 Debouncer

```
`ifndef INCLUDED_A7BDC842_279E_4ABD_BC4A_B9583B395075
    `define INCLUDED_A7BDC842_279E_4ABD_BC4A_B9583B395075

    // Switch Debounce Module
    // use your system clock for the clock input
    // to produce a synchronous, debounced output
    // delay is .01 sec with a 27Mhz clock
    module Debouncer #(parameter delay=270000, countBits=19)
    (
        input       clock,
        input       reset,
        input       noisy,

        output reg  clean
    );

        reg [countBits-1:0] count;
        reg                 _new;    //the name _new is used so that it
                                     //does not clash with the keyword
                                     //C++ when using Verilator to confirm
                                     //synthesizability.
        always @(posedge clock)
            if (reset)
                begin
                    count <= 0;
                    _new  <= noisy;
                    clean <= noisy;
                end
            else if (noisy != _new)
                begin
                    _new  <= noisy;
                    count <= 0;
                end
            else if (count == (delay-1))
                clean <= _new;
            else
                count <= count+1;

    endmodule

`endif
//newline required
```

13.2 Enabler

```
`ifndef INCLUDE_AB6113A9_1AFC_4699_A6BD_3C5E503EFCC8
    `define INCLUDE_AB6113A9_1AFC_4699_A6BD_3C5E503EFCC8

    //////////////////////////////////////////////////////////
    ////
    //// This modules selects certain cycles of an input clock.
    //// The module can be configured to signal every certain
    //// number of clock cycles, thereby creating a kind of clock.
    ////
    //// However, the signal produced is only HIGH for one cycle
    //// of the input clock and LOW for the unselected cycles
    //// of the input clock.
    ////
    //// These following parameters can be set:
    ////     1. cyclesPerEnable : the number of cycles to place between enable signals;
    ////                          it must be at least 1.
    ////
    ////     2. countBits       : the number of bits to use for storing the cycle count,
    ////                          which is a number as big as (cyclesPerEnable-1).
    ////
    //// The default parameter values create a 1Hz enable with a 27Mhz input clock.
    ////
    //// NOTES: Perhaps a better name would be CycleSelector.
    ////
    //////////////////////////////////////////////////////////
    module Enabler
    (
        input      clk,     //HIGH: triggers logic; defaults are set for a 27 Mhz clock
        input      reset,   //HIGH: asynchronous; sets the enable to LOW, and begins counting from the current cycle.
        output reg enable   //HIGH: A cycle is being selected
    );
        parameter countBits       = 25;         //big enough for cyclesPerEnable-1
        parameter cyclesPerEnable = 27000000;   //Must be at least 1

        reg[countBits-1:0] count;

        always @ (posedge clk or posedge reset)
            if (reset)
                begin
                    count  <= 0;
                    enable <= 0;
                end
            else if (count == (cyclesPerEnable-1)) //Was the last cycle the last second to last in our count?
                begin                              //That is, is this cycle the signal to turn on the enable?
                    count  <= 0;
                    enable <= 1;
                end
            else                                   //Keep counting then.
                begin
                    count  <= count + 1;
                    enable <= 0;
                end
    endmodule

`endif
//newline required
```

13.3 EnableCounter

```
`ifndef INCLUDE_D141CED5_ED8D_4135_842F_15141597FB6B
    `define INCLUDE_D141CED5_ED8D_4135_842F_15141597FB6B

    //////////////////////////////////////////////////////////
```

```
    ////
    //// This counts high enable bits every clock cycle,
    //// and then terminates after a given number (enableCount)
    //// has been reached. It signals that it has stopped (stop).
    ////
    //// Upon reset, the enableCount is noted.
    ////
    //// The enable bit must be synchronized to the clock.
    ////
    //// Naturally, if (enableCount) is zero, (stop) is constantly HIGH.
    ////
    //// This can be used as a timer, if the enable bits are high at
    //// regular intervals.
    ////
    ////
    //////////////////////////////////////////////////////////
    module EnableCounter #(parameter countBits=4)
    (
        input                 clk,            //HIGH: triggers logic
        input                 reset,          //HIGH: sets the counter counting again.
        input                 enable,         //HIGH: An enable should be counted (see Enabler.v)
        input [countBits-1:0] enableCount,    //the number of enables to count before stopping;

        output reg            stop            //HIGH: the counter has finished counting.
    );

        reg [countBits-1:0] enablesLeft;                  //It is better to count down,
                                                          //as it requires less logic.
        always @ (posedge clk    or    posedge reset)
            begin
                stop <= 0;                                //Pulse stop one cycle

                if (reset)
                    begin
                        enablesLeft <= enableCount;

                        if (enableCount == 0)             //enableCount==0 implies counting finished,
                            stop <= 1;                    //but only pulse for one cycle.
                    end
                else if (enable)                          //It's been one time unit
                    begin
                        if (enablesLeft == 1)             //Is this the last counted enable?
                            stop <= 1;

                        if (enablesLeft != 0)
                            enablesLeft <= enablesLeft-1; //Count down; less logic than counting up.
                    end
            end

    endmodule

`endif
//newline required
```

13.4 LevelToPulse

```
module `OPERATOR_NAME
(
    input clk,

    input reset,
    input resetValue,

    input level,

    output pulse
```

```
);

    reg levelOld;

    always @ (posedge clk)
        if (reset)
            levelOld <= resetValue;
        else
            levelOld <= level;

    assign pulse = `OPERATOR;

endmodule
```

13.4.1 LevelToPulseNeg

```
`ifndef INCLUDE_C2811789_0E75_4C25_934E_A177F1A26F4A
    `define INCLUDE_C2811789_0E75_4C25_934E_A177F1A26F4A

    `define OPERATOR_NAME LevelToPulseNeg
    `define OPERATOR      !level & levelOld

    `include "src/synthesis/util/LevelToPulse.v"

`endif
//newline required
```

13.4.2 LevelToPulsePos

```
`ifndef INCLUDE_DF3A33A2_2E31_4DD8_B0C8_64A1421F8E8F
    `define INCLUDE_DF3A33A2_2E31_4DD8_B0C8_64A1421F8E8F

    `define OPERATOR_NAME LevelToPulsePos
    `define OPERATOR      level & !levelOld

    `include "src/synthesis/util/LevelToPulse.v"

`endif
//newline required
```

13.5 Delay

```
`ifndef INCLUDE_B16C669E_89EC_46EB_98EB_11771A111A72
    `define INCLUDE_B16C669E_89EC_46EB_98EB_11771A111A72

    ////////////////////////////////////////////////////////////////////////////////
    ////
    //// Delay
    //// This module implements delays in general
    //// using these parameters:
    ////     (1) delay: the number of cycles to delay
    ////     (2) width: the size of the sigal.
    ////
    ////////////////////////////////////////////////////////////////////////////////


    module Delay
    #(parameter
```

```
        delay  = 1,
        width  = 1
    )
    (
        input              clk,

        input  [width-1:0] signal,

        output [width-1:0] signalDelayed
    );

        reg [width-1:0] pipeline [delay-1:0];

        integer i;
        always @ (posedge clk)
            begin
                for (i=delay-1; i > 0; i = i-1)
                    pipeline[i] <= pipeline[i-1];

                pipeline[0] <= signal;
            end

        assign signalDelayed = pipeline[delay-1];

    endmodule

`endif
//newline required
```

13.5.1 TestDelay

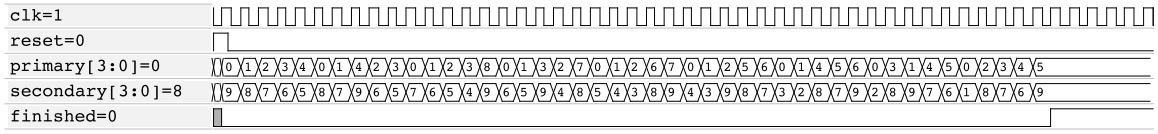

Figure 15:

```
`include "src/synthesis/util/Delay.v"
`include "src/simulation/Clock.v"

module TestDelay;

    wire clk;
    Clock clock(0, clk);

    reg  [1:0] signal;
    wire [1:0] signalDelayed;

    Delay #(.delay(2), .width(2)) delay
    (
        clk,
        signal,
        signalDelayed
    );

    initial
        begin
            $dumpvars;
            $monitor("cycle %d: %b", clock.cycle, signalDelayed);
        end
```

```
    always @ (negedge clk)
        case (clock.cycle)
                        1: signal <= 2'd3;
                        2: signal <= 2'd0;
            delay.delay+3: $finish;
        endcase

endmodule
```

13.6 FIFO

```
`ifndef INCLUDE_8A884CAD_F04F_4CAB_B8EC_419CBFEA1942
    `define INCLUDE_8A884CAD_F04F_4CAB_B8EC_419CBFEA1942

    module FIFO
    #(parameter
        width     = 21,

        depth     =  8,  // Must be a power of 2.
        depthBits =  3
    )(
        input              clk,
        input              reset,

        input  [width-1:0] dataToStore,
        input              store,
        input              read,

        output reg         full,
        output             available,
        output [width-1:0] dataToRead
    );

        reg  [width-1:0] storage [depth-1:0];

        reg  [depthBits-1:0] ptrRead;
        reg  [depthBits-1:0] ptrWrite;
        wire [depthBits-1:0] ptrWriteNext = ptrWrite+1;

        wire fullNext = (ptrWriteNext == ptrRead);

        always @ (posedge clk)
            if (reset)
                begin
                    full       <= 0;
                    ptrWrite   <= 0;
                    ptrRead    <= 0;
                    storage[0] <= 0;
                end
            else
                begin
                    if (store && !full)
                        begin
                            storage[ptrWrite] <= dataToStore;
                            ptrWrite          <= ptrWriteNext;

                            full <= fullNext;
                        end

                    if (read && available)
                        begin
                            ptrRead <= ptrRead+1;
                            full    <= 0;
                        end
                end
```

```
        assign available  = (ptrWrite != ptrRead) || full,
               dataToRead = storage[ptrRead];

    endmodule

‘endif
//newline required
```

13.6.1 TestFIFO

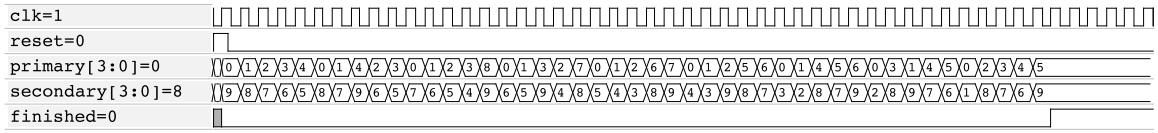

Figure 16:

```
‘include "src/synthesis/util/FIFO.v"
‘include "src/simulation/Clock.v"

module TestFIFO;

    wire clk;
    Clock #(.timeClock(5)) clock(0, clk);

    reg  reset = 1,
         store = 1,
         read  = 0;

    wire full,
         available;

    wire [20:0] dataToRead;
    wire [20:0] #2 dataToStore = clock.cycle;

    FIFO fifo
    (
        clk,
        reset,

        dataToStore,
        store,
        read,

        full,
        available,
        dataToRead
    );

    initial
        $dumpvars;

    always @ (negedge clk)
        case (clock.cycle)
             1: #2 reset <= 0;
             9: #2 store <= 0;
            10: #2 read  <= 1;
            19: #2 read  <= 0;
            20: #2 store <= 1;
            28: #2 store <= 0;
            29: #2 read  <= 1;
```

```
            40: #2 read  <= 0;
            41: #2 store <= 1;
            42: #2 read  <= 1;
            55: #2 $finish;
        endcase

endmodule
```