Bitwise

Release 0.1.1

May 25, 2019

Contents

CHAPTER 1

Arithmetic

- *Adder4*
- *Adder8*
- *Adder16*
- *AdderSubtractor4*
- *AdderSubtractor8*
- *AdderSubtractor16*
- *FullAdder*
- *HalfAdder*
- *Multiplier2*
- *Multiplier4*
- *Multiplier8*

CHAPTER 2

Gate

- *ANDGate2*
- *ANDGate3*
- *ANDGate4*
- *Buffer*
- *IMPLYGate*
- *NANDGate2*
- *NANDGate3*
- *NANDGate4*
- *NORGate2*
- *NORGate3*
- *NORGate4*
- *NOTGate*
- *ORGate2*
- *ORGate3*
- *ORGate4*
- *XNORGate2*
- *XORGate2*

CHAPTER 3

Logic

- *BitwiseAND4*
- *BitwiseAND8*
- *BitwiseAND16*
- *BitwiseNAND4*
- *BitwiseNAND8*
- *BitwiseNAND16*
- *BitwiseNOR4*
- *BitwiseNOR8*
- *BitwiseNOR16*
- *BitwiseNOT4*
- *BitwiseNOT8*
- *BitwiseOR4*
- *BitwiseOR8*
- *BitwiseOR16*
- *BitwiseNOT16*
- *BitwiseXNOR4*
- *BitwiseXNOR8*
- *BitwiseXNOR16*
- *BitwiseXOR4*
- *BitwiseXOR8*
- *BitwiseXOR16*
- *Comparator3*

- *Comparator7*
- *Comparator15*
- *ParityChecker4*
- *ParityChecker8*
- *ParityChecker16*
- *ParityGenerator4*
- *ParityGenerator8*
- *ParityGenerator16*

CHAPTER 4

Processor

- *ArithmeticLogicUnit*
- *ConditionCodeFlags*
- *ProgramCounter*
- *StackPointer*

CHAPTER 5

Signal

- *ControlledInverter4*
- *ControlledInverter8*
- *ControlledInverter16*
- *Decoder1Of4*
- *Decoder1Of8*
- *Decoder1Of16*
- *Demultiplexer1To2*
- *Demultiplexer1To4*
- *Demultiplexer1To8*
- *Demultiplexer1To16*
- *Encoder4To2*
- *Encoder8To3*
- *Encoder16To4*
- *Multiplexer2To1*
- *Multiplexer4To1*
- *Multiplexer8To1*
- *Multiplexer16To1*
- *SevenSegmentConverter*
- *SevenSegmentConverterDual*
- *SevenSegmentConverterQuad*

CHAPTER 6

State

- *DownCounterMod4*
- *DownCounterMod8*
- *DownCounterMod16*
- *ParallelToSerialConverter4To1*
- *ParallelToSerialConverter8To1*
- *ParallelToSerialConverter16To1*
- *RingCounter4*
- *RingCounter8*
- *RingCounter16*
- *SerialToParallelConverter1To4*
- *SerialToParallelConverter1To8*
- *SerialToParallelConverter1To16*
- *ShiftRegister4*
- *ShiftRegister8*
- *ShiftRegister16*
- *UpCounterMod4*
- *UpCounterMod8*
- *UpCounterMod16*

CHAPTER 7

Storage

- *DFlipFlop*
- *DFlipFlopPresetClear*
- *GatedDLatch*
- *GatedSRLatch*
- *JKFlipFlop*
- *JKFlipFlopPresetClear*
- *RAM16x4*
- *RAM256x4*
- *RAM65536x4*
- *RAM16x8*
- *RAM256x8*
- *RAM65536x8*
- *RAM16x16*
- *RAM256x16*
- *RAM65536x16*
- *Register4*
- *Register8*
- *Register16*
- *SRLatch*
- *TFlipFlop*
- *TFlipFlopPresetClear*

CHAPTER 8

Wire

- *BufferBus4*
- *BufferBus8*
- *BufferBus16*
- *Bus4*
- *Bus8*
- *Bus16*
- *BusSevenSegmentDisplay*
- *TristateBuffer*
- *Wire*

CHAPTER 9

Arithmetic

9.1 Adder4

9.1.1 Class `bw.arithmetic.Adder4`

Defined in bitwise/arithmetic/ADD.py.

4-bit adder.

9.1.2 __init__

```
__init__(
    carry_in,
    a_bus,
    b_bus,
    carry_out,
    sum_bus
)
```

Construct a new 4-bit adder.

Args:

- `carry_in`: An object of type `Wire`. The carry-in to the adder.
- `a_bus`: An object of type `Bus4`. The first addend. `a_bus[0]` and `a_bus[3]` are the most and least significant bit, respectively.
- `b_bus`: An object of type `Bus4`. The second addend. `b_bus[0]` and `b_bus[3]` are the most and least significant bit, respectively.

- `carry_out`: An object of type `Wire`. The carry-out of the adder.
- `sum_bus`: An object of type `Bus4`. The sum of the two addends. `sum_bus[0]` and `sum_bus[3]` are the most and least significant bit, respectively.

Raises:

- `TypeError`: If either `a_bus`, `b_bus`, or `sum_bus` is not a bus of width 4.

9.1.3 __str__

Print out the wire values of the 4-bit adder.

```
carry_in: 0
a_bus: (0, 0, 0, 0)
b_bus: (0, 0, 0, 0)
carry_out: 0
sum_bus: (0, 0, 0, 0)
```

9.1.4 __call__

```
__call__(
    carry_in=None,
    a_bus=None,
    b_bus=None,
    carry_out=None,
    sum_bus=None
)
```

Force specific values on the wires of the 4-bit adder.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

9.2 Adder8

9.2.1 Class `bw.arithmetic.Adder8`

Defined in bitwise/arithmetic/ADD.py.

8-bit adder.

9.2.2 __init__

```
__init__(
    carry_in,
    a_bus,
    b_bus,
    carry_out,
```

(continues on next page)

(continued from previous page)

```
    sum_bus
)
```

Construct a new 8-bit adder.

Args:

- `carry_in`: An object of type `Wire`. The carry-in to the adder.
- `a_bus`: An object of type `Bus8`. The first addend. `a_bus[0]` and `a_bus[7]` are the most and least significant bit, respectively.
- `b_bus`: An object of type `Bus8`. The second addend. `b_bus[0]` and `b_bus[7]` are the most and least significant bit, respectively.
- `carry_out`: An object of type `Wire`. The carry-out of the adder.
- `sum_bus`: An object of type `Bus8`. The sum of the two addends. `sum_bus[0]` and `sum_bus[7]` are the most and least significant bit, respectively.

Raises:

- `TypeError`: If either `a_bus`, `b_bus`, or `sum_bus` is not a bus of width 8.

9.2.3 __str__

Print out the wire values of the 8-bit adder.

```
carry_in: 0
a_bus: (0, 0, 0, 0, 0, 0, 0, 0)
b_bus: (0, 0, 0, 0, 0, 0, 0, 0)
carry_out: 0
sum_bus: (0, 0, 0, 0, 0, 0, 0, 0)
```

9.2.4 __call__

```
__call__(
    carry_in=None,
    a_bus=None,
    b_bus=None,
    carry_out=None,
    sum_bus=None
)
```

Force specific values on the wires of the 8-bit adder.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

9.3 Adder16

9.3.1 Class `bw.arithmetic.Adder16`

Defined in bitwise/arithmetic/ADD.py.

16-bit adder.

9.3.2 __init__

```
__init__(
    carry_in,
    a_bus,
    b_bus,
    carry_out,
    sum_bus
)
```

Construct a new 16-bit adder.

Args:

- `carry_in`: An object of type `Wire`. The carry-in to the adder.
- `a_bus`: An object of type `Bus16`. The first addend. `a_bus[0]` and `a_bus[15]` are the most and least significant bit, respectively.
- `b_bus`: An object of type `Bus16`. The second addend. `b_bus[0]` and `b_bus[15]` are the most and least significant bit, respectively.
- `carry_out`: An object of type `Wire`. The carry-out of the adder.
- `sum_bus`: An object of type `Bus16`. The sum of the two addends. `sum_bus[0]` and `sum_bus[15]` are the most and least significant bit, respectively.

Raises:

- `TypeError`: If either `a_bus`, `b_bus`, or `sum_bus` is not a bus of width 16.

9.3.3 __str__

Print out the wire values of the 16-bit adder.

```
carry_in: 0
a_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
b_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
carry_out: 0
sum_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
```

9.3.4 __call__

```
__call__(
    carry_in=None,
    a_bus=None,
    b_bus=None,
    carry_out=None,
    sum_bus=None
)
```

Force specific values on the wires of the 16-bit adder.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

9.4 AdderSubtractor4

9.4.1 Class `bw.arithmetic.AdderSubtractor4`

Defined in bitwise/arithmetic/ADD_SUB.py.

4-bit adder-subtractor.

9.4.2 __init__

```
__init__(
    add_subtract,
    a_bus,
    b_bus,
    overflow,
    carry_out,
    sum_bus
)
```

Construct a new 4-bit adder-subtractor.

Args:

- `add_subtract`: An object of type `Wire`. Indicates the operation to carry out - 0 for addition, 1 for subtraction.
- `a_bus`: An object of type `Bus4`. The first addend, or the minuend. `a_bus[0]` and `a_bus[3]` are the most and least significant bit, respectively. `a_bus[0]` is the sign bit in subtraction operations.
- `b_bus`: An object of type `Bus4`. The second addend, or the subtrahend. `b_bus[0]` and `b_bus[3]` are the most and least significant bit, respectively. `b_bus[0]` is the sign bit in subtraction operations.
- `overflow`: An object of type `Wire`. The overflow indicator of the subtractor.
- `carry_out`: An object of type `Wire`. The carry-out of the adder.
- `sum_bus`: An object of type `Bus4`. The sum of the two addends, or the difference between the minuend and the subtrahend. `sum_bus[0]` and `sum_bus[3]` are the most and least significant bit, respectively. `sum_bus[0]` is the sign bit in subtraction operations.

Raises:

- `TypeError`: If either `a_bus`, `b_bus`, or `sum_bus` is not a bus of width 4.

9.4.3 __str__

Print out the wire values of the 4-bit adder-subtractor.

```
add_subtract: 0
a_bus: (0, 0, 0, 0)
b_bus: (0, 0, 0, 0)
overflow: 0
carry_out: 0
sum_bus: (0, 0, 0, 0)
```

9.4.4 __call__

```
__call__(
    add_subtract=None,
    a_bus=None,
    b_bus=None,
    overflow=None,
    carry_out=None,
    sum_bus=None
)
```

Force specific values on the wires of the 4-bit adder-subtractor.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

9.5 AdderSubtractor8

9.5.1 Class `bw.arithmetic.AdderSubtractor8`

Defined in bitwise/arithmetic/ADD_SUB.py.

8-bit adder-subtractor.

9.5.2 __init__

```
__init__(
    add_subtract,
    a_bus,
    b_bus,
    overflow,
    carry_out,
    sum_bus
)
```

Construct a new 8-bit adder-subtractor.

Args:

- `add_subtract`: An object of type `Wire`. Indicates the operation to carry out - 0 for addition, 1 for subtraction.
- `a_bus`: An object of type `Bus8`. The first addend, or the minuend. `a_bus[0]` and `a_bus[7]` are the most and least significant bit, respectively. `a_bus[0]` is the sign bit in subtraction operations.
- `b_bus`: An object of type `Bus8`. The second addend, or the subtrahend. `b_bus[0]` and `b_bus[7]` are the most and least significant bit, respectively. `b_bus[0]` is the sign bit in subtraction operations.
- `overflow`: An object of type `Wire`. The overflow indicator of the subtractor.
- `carry_out`: An object of type `Wire`. The carry-out of the adder.
- `sum_bus`: An object of type `Bus8`. The sum of the two addends, or the difference between the minuend and the subtrahend. `sum_bus[0]` and `sum_bus[7]` are the most and least significant bit, respectively. `sum_bus[0]` is the sign bit in subtraction operations.

Raises:

- `TypeError`: If either `a_bus`, `b_bus`, or `sum_bus` is not a bus of width 8.

9.5.3 __str__

Print out the wire values of the 8-bit adder-subtractor.

```
add_subtract: 0
a_bus: (0, 0, 0, 0, 0, 0, 0, 0)
b_bus: (0, 0, 0, 0, 0, 0, 0, 0)
overflow: 0
carry_out: 0
sum_bus: (0, 0, 0, 0, 0, 0, 0, 0)
```

9.5.4 __call__

```
__call__(
    add_subtract=None,
    a_bus=None,
    b_bus=None,
    overflow=None,
    carry_out=None,
    sum_bus=None
)
```

Force specific values on the wires of the 8-bit adder-subtractor.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

9.6 AdderSubtractor16

9.6.1 Class `bw.arithmetic.AdderSubtractor16`

Defined in bitwise/arithmetic/ADD_SUB.py.

16-bit adder-subtractor.

9.6.2 __init__

```
__init__(
    add_subtract,
    a_bus,
    b_bus,
    overflow,
    carry_out,
    sum_bus
)
```

Construct a new 16-bit adder-subtractor.

Args:

- `add_subtract`: An object of type `Wire`. Indicates the operation to carry out - 0 for addition, 1 for subtraction.
- `a_bus`: An object of type `Bus16`. The first addend, or the minuend. `a_bus[0]` and `a_bus[15]` are the most and least significant bit, respectively. `a_bus[0]` is the sign bit in subtraction operations.
- `b_bus`: An object of type `Bus16`. The second addend, or the subtrahend. `b_bus[0]` and `b_bus[15]` are the most and least significant bit, respectively. `b_bus[0]` is the sign bit in subtraction operations.
- `overflow`: An object of type `Wire`. The overflow indicator of the subtractor.
- `carry_out`: An object of type `Wire`. The carry-out of the adder.
- `sum_bus`: An object of type `Bus16`. The sum of the two addends, or the difference between the minuend and the subtrahend. `sum_bus[0]` and `sum_bus[15]` are the most and least significant bit, respectively. `sum_bus[0]` is the sign bit in subtraction operations.

Raises:

- `TypeError`: If either `a_bus`, `b_bus`, or `sum_bus` is not a bus of width 16.

9.6.3 __str__

Print out the wire values of the 16-bit adder-subtractor.

```
add_subtract: 0
a_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
b_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
overflow: 0
carry_out: 0
sum_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
```

9.6.4 __call__

```
__call__(
    add_subtract=None,
    a_bus=None,
    b_bus=None,
    overflow=None,
    carry_out=None,
    sum_bus=None
)
```

Force specific values on the wires of the 16-bit adder-subtractor.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

9.7 FullAdder

9.7.1 Class `bw.arithmetic.FullAdder`

Defined in bitwise/arithmetic/ADD.py.

Full adder.

9.7.2 __init__

```
__init__(
    carry_in,
    a,
    b,
    carry_out,
    sum
)
```

Construct a new full adder.

Args:

- `carry_in`: An object of type `Wire`. The carry-in to the adder.
- `a`: An object of type `Wire`. The first addend.
- `b`: An object of type `Wire`. The second addend.
- `carry_out`: An object of type `Wire`. The carry-out of the adder.

- `sum`: An object of type `Wire`. The sum of the two addends.

9.7.3 __str__

Print out the wire values of the full adder.

```
carry_in: 0
a: 0
b: 0
carry_out: 0
sum: 0
```

9.7.4 __call__

```
__call__(
    carry_in=None,
    a=None,
    b=None,
    carry_out=None,
    sum=None
)
```

Force specific values on the wires of the full adder.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

9.8 HalfAdder

9.8.1 Class `bw.arithmetic.HalfAdder`

Defined in bitwise/arithmetic/ADD.py.

Half adder.

9.8.2 __init__

```
__init__(
    a,
    b,
    carry_out,
    sum
)
```

Construct a new half adder.

Args:

- `a`: An object of type `Wire`. The first addend.

- `b`: An object of type `Wire`. The second addend.
- `carry_out`: An object of type `Wire`. The carry-out of the adder.
- `sum`: An object of type `Wire`. The sum of the two addends.

9.8.3 __str__

Print out the wire values of the half adder.

```
a: 0
b: 0
carry_out: 0
sum: 0
```

9.8.4 __call__

```
__call__(
    a=None,
    b=None,
    carry_out=None,
    sum=None
)
```

Force specific values on the wires of the half adder.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

9.9 Multiplier2

9.9.1 Class `bw.arithmetic.Multiplier2`

Defined in bitwise/arithmetic/MULT.py.

2-bit unsigned array multiplier.

9.9.2 __init__

```
__init__(
    a_1,
    a_2,
    b_1,
    b_2,
    product_bus
)
```

Construct a new 2-bit unsigned multiplier.

Args:

- `a_1`: An object of type `Wire`. The most significant bit of the multiplicand.
- `a_2`: An object of type `Wire`. The least significant bit of the multiplicand.
- `b_1`: An object of type `Wire`. The most significant bit of the multiplier.
- `b_2`: An object of type `Wire`. The least significant bit of the multiplier.
- `product_bus`: An object of type `Bus4`. The product. `product_bus[0]` and `product_bus[3]` are the most and least significant bit, respectively.

Raises:

- `TypeError`: If `product_bus` is not a bus of width 4.

9.9.3 __str__

Print out the wire values of the 2-bit unsigned multiplier.

```
a_1: 0
a_2: 0
b_1: 0
b_2: 0
product_bus: (0, 0, 0, 0)
```

9.9.4 __call__

```
__call__(
    a_1=None,
    a_2=None,
    b_1=None,
    b_2=None,
    product_bus=None
)
```

Force specific values on the wires of the 2-bit unsigned multiplier.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

9.10 Multiplier4

9.10.1 Class `bw.arithmetic.Multiplier4`

Defined in bitwise/arithmetic/MULT.py.

4-bit unsigned array multiplier.

9.10.2 __init__

```
__init__(
    a_bus,
    b_bus,
    product_bus
)
```

Construct a new 4-bit unsigned multiplier.

Args:

- `a_bus`: An object of type `Bus4`. The multiplicand. `a_bus[0]` and `a_bus[3]` are the most and least significant bit, respectively.
- `b_bus`: An object of type `Bus4`. The multiplier. `b_bus[0]` and `b_bus[3]` are the most and least significant bit, respectively.
- `product_bus`: An object of type `Bus8`. The product. `product_bus[0]` and `product_bus[7]` are the most and least significant bit, respectively.

Raises:

- `TypeError`: If either `a_bus` or `b_bus` is not a bus of width 4, or if `product_bus` is not a bus of width 8.

9.10.3 __str__

Print out the wire values of the 4-bit unsigned multiplier.

```
a_bus: (0, 0, 0, 0)
b_bus: (0, 0, 0, 0)
product_bus: (0, 0, 0, 0, 0, 0, 0, 0)
```

9.10.4 __call__

```
__call__(
    a_bus=None,
    b_bus=None,
    product_bus=None
)
```

Force specific values on the wires of the 4-bit unsigned multiplier.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

9.11 Multiplier8

9.11.1 Class `bw.arithmetic.Multiplier8`

Defined in bitwise/arithmetic/MULT.py.

8-bit unsigned array multiplier.

9.11.2 __init__

```
__init__(
    a_bus,
    b_bus,
    product_bus
)
```

Construct a new 8-bit unsigned multiplier.

Args:

- `a_bus`: An object of type `Bus8`. The multiplicand. `a_bus[0]` and `a_bus[7]` are the most and least significant bit, respectively.
- `b_bus`: An object of type `Bus8`. The multiplier. `b_bus[0]` and `b_bus[7]` are the most and least significant bit, respectively.
- `product_bus`: An object of type `Bus16`. The product. `product_bus[0]` and `product_bus[15]` are the most and least significant bit, respectively.

Raises:

- `TypeError`: If either `a_bus` or `b_bus` is not a bus of width 8, or if `product_bus` is not a bus of width 16.

9.11.3 __str__

Print out the wire values of the 16-bit unsigned multiplier.

```
a_bus: (0, 0, 0, 0, 0, 0, 0, 0)
b_bus: (0, 0, 0, 0, 0, 0, 0, 0)
product_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
```

9.11.4 __call__

```
__call__(
    a_bus=None,
    b_bus=None,
    product_bus=None
)
```

Force specific values on the wires of the 16-bit unsigned multiplier.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

CHAPTER 10

Changelog

10.1 1.0 - 2019-05-25

10.1.1 Added

- **RAM modules to storage subpackage**
  - `RAM16x4`
  - `RAM256x4`
  - `RAM65536x4`
  - `RAM16x8`
  - `RAM256x8`
  - `RAM65536x8`
  - `RAM16x16`
  - `RAM256x16`
  - `RAM65536x16`
- Condition code flag flip-flops module to processor subpackage
- Stack pointer (SP) and program counter (PC) modules to processor subpackage
- Overloaded `__str__()` methods to all modules
- Overloaded `__call__()` methods to all modules

10.1.2 Changed

- Modified `__init__()` methods to allow all modules to be initialized with keyword arguments

10.1.3 Removed

- 10-bit bus module from processor subpackage
- Processor from processor subpackage

10.2 0.3 - 2019-01-26

10.2.1 Added

- Processor, arithmetic-logic unit, and 10-bit bus modules to processor subpackage
- **Bitwise logic operations to logic subpackage**
  - Bitwise AND, NAND, NOR, NOT, OR, XNOR, and XOR operations for 4-, 8-, and 16- bit inputs
- **Multiplier classes to arithmetic subpackage**
  - `Multiplier2`
  - `Multiplier4`
  - `Multiplier8`

10.2.2 Changed

- **Fixed enable inputs in the following modules to no longer act as a positive clock edge**
  - `ParallelToSerialConverter4To1`
  - `ParallelToSerialConverter8To1`
  - `ParallelToSerialConverter16To1`
  - `ShiftRegister4`
  - `ShiftRegister8`
  - `ShiftRegister16`
  - `SerialToParallelConverter1To4`
  - `SerialToParallelConverter1To8`
  - `SerialToParallelConverter1To16`
- Added enable inputs to registers in `storage.REG`
- Changed ordering of parameters to `TristateBuffer`

10.3 v0.2 - 2018-11-08

10.3.1 Added

- Tri-state buffer to wire subpackage
- **Ring counter classes to state subpackage**
  - `RingCounter4`

    - `RingCounter8`
    - `RingCounter16`
- **Up- and down-counter classes to state subpackage**
    - `UpCounterMod4`
    - `UpCounterMod8`
    - `UpCounterMod16`
    - `DownCounterMod4`
    - `DownCounterMod8`
    - `DownCounterMod16`
- `IMPLY` logic gate to gate subpackage
- **Shift register classes to state subpackage**
    - `ShiftRegister4`
    - `ShiftRegister8`
    - `ShiftRegister16`
- **Parallel-to-serial converter classes to state subpackage**
    - `ParallelToSerialConverter4To1`
    - `ParallelToSerialConverter8To1`
    - `ParallelToSerialConverter16To1`
- **Serial-to-parallel converter classes to state subpackage**
    - `SerialToParallelConverter1To4`
    - `SerialToParallelConverter1To8`
    - `SerialToParallelConverter1To16`
- `Buffer` class to gate subpackage

10.3.2 Changed

- Added `__getitem__()` and `__len__()` methods to `Bus4`, `Bus8`, `Bus16`, and `BusSevenSegmentDisplay` classes
- Rewrote docstrings for all existing classes
- Misc improvements

10.4 v0.1.1 - 2018-10-17

10.4.1 Added

- **Storage register classes to storage subpackage**
    - `Register4`
    - `Register8`

- `Register16`

10.4.2 Changed

- Misc improvements

CHAPTER 11

Gate

11.1 ANDGate2

11.1.1 Class `bw.gate.ANDGate2`

Defined in bitwise/gate/AND.py.

Two-input AND gate.

11.1.2 __init__

```
__init__(
    input_1,
    input_2,
    output
)
```

Construct a new two-input AND gate.

Args:

- `input_1`: An object of type `Wire`. The first input to the AND gate.
- `input_2`: An object of type `Wire`. The second input to the AND gate.
- `output`: An object of type `Wire`. The output of the AND gate.

11.1.3 __str__

Print out the wire values of the AND gate.

```
input_1: 0
input_2: 0
output: 0
```

11.1.4 __call__

```
__call__(
    input_1=None,
    input_2=None,
    output=None
)
```

Force specific values on the wires of the AND gate.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

11.2 ANDGate3

11.2.1 Class `bw.gate.ANDGate3`

Defined in bitwise/gate/AND.py.

Three-input AND gate.

11.2.2 __init__

```
__init__(
    input_1,
    input_2,
    input_3,
    output
)
```

Construct a new three-input AND gate.

Args:

- `input_1`: An object of type `Wire`. The first input to the AND gate.
- `input_2`: An object of type `Wire`. The second input to the AND gate.
- `input_3`: An object of type `Wire`. The third input to the AND gate.
- `output`: An object of type `Wire`. The output of the AND gate.

11.2.3 __str__

Print out the wire values of the AND gate.

```
input_1: 0
input_2: 0
input_3: 0
output: 0
```

11.2.4 __call__

```
__call__(
    input_1=None,
    input_2=None,
    input_3=None,
    output=None
)
```

Force specific values on the wires of the AND gate.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

11.3 ANDGate4

11.3.1 Class `bw.gate.ANDGate4`

Defined in bitwise/gate/AND.py.

Four-input AND gate.

11.3.2 __init__

```
__init__(
    input_1,
    input_2,
    input_3,
    input_4,
    output
)
```

Construct a new four-input AND gate.

Args:

- `input_1`: An object of type `Wire`. The first input to the AND gate.
- `input_2`: An object of type `Wire`. The second input to the AND gate.
- `input_3`: An object of type `Wire`. The third input to the AND gate.
- `input_4`: An object of type `Wire`. The fourth input to the AND gate.

- `output`: An object of type `Wire`. The output of the AND gate.

11.3.3 __str__

Print out the wire values of the AND gate.

```
input_1: 0
input_2: 0
input_3: 0
input_4: 0
output: 0
```

11.3.4 __call__

```
__call__(
    input_1=None,
    input_2=None,
    input_3=None,
    input_4=None,
    output=None
)
```

Force specific values on the wires of the AND gate.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

11.4 Buffer

11.4.1 Class `bw.gate.Buffer`

Defined in bitwise/gate/BUF.py.

Digital buffer.

11.4.2 __init__

```
__init__(
    input,
    output
)
```

Construct a new buffer.

Args:

- `input`: An object of type `Wire`. The input to the buffer.
- `output`: An object of type `Wire`. The output of the buffer.

11.4.3 __str__

Print out the wire values of the buffer.

```
input: 0
output: 0
```

11.4.4 __call__

```
__call__(
    input=None,
    output=None
)
```

Force specific values on the wires of the buffer.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

11.5 IMPLYGate

11.5.1 Class `bw.gate.IMPLYGate`

Defined in bitwise/gate/IMPLY.py.

IMPLY gate.

11.5.2 __init__

```
__init__(
    input_1,
    input_2,
    output
)
```

Construct a new IMPLY gate.

Args:

- `input_1`: An object of type `Wire`. The first input to the IMPLY gate.
- `input_2`: An object of type `Wire`. The second input to the IMPLY gate.
- `output`: An object of type `Wire`. The output of the IMPLY gate.

11.5.3 __str__

Print out the wire values of the IMPLY gate.

```
input_1: 0
input_2: 0
output: 0
```

11.5.4 __call__

```
__call__(
    input_1=None,
    input_2=None,
    output=None
)
```

Force specific values on the wires of the IMPLY gate.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

11.6 NANDGate2

11.6.1 Class `bw.gate.NANDGate2`

Defined in bitwise/gate/NAND.py.

Two-input NAND gate.

11.6.2 __init__

```
__init__(
    input_1,
    input_2,
    output
)
```

Construct a new two-input NAND gate.

Args:

- `input_1`: An object of type `Wire`. The first input to the NAND gate.
- `input_2`: An object of type `Wire`. The second input to the NAND gate.
- `output`: An object of type `Wire`. The output of the NAND gate.

11.6.3 __str__

Print out the wire values of the NAND gate.

```
input_1: 0
input_2: 0
output: 0
```

11.6.4 __call__

```
__call__(
    input_1=None,
    input_2=None,
    output=None
)
```

Force specific values on the wires of the NAND gate.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

11.7 NANDGate3

11.7.1 Class `bw.gate.NANDGate3`

Defined in bitwise/gate/NAND.py.

Three-input NAND gate.

11.7.2 __init__

```
__init__(
    input_1,
    input_2,
    input_3,
    output
)
```

Construct a new three-input NAND gate.

Args:

- `input_1`: An object of type `Wire`. The first input to the NAND gate.
- `input_2`: An object of type `Wire`. The second input to the NAND gate.
- `input_3`: An object of type `Wire`. The third input to the NAND gate.
- `output`: An object of type `Wire`. The output of the NAND gate.

11.7.3 __str__

Print out the wire values of the NAND gate.

```
input_1: 0
input_2: 0
input_3: 0
output: 0
```

11.7.4 __call__

```
__call__(
    input_1=None,
    input_2=None,
    input_3=None,
    output=None
)
```

Force specific values on the wires of the NAND gate.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

11.8 NANDGate4

11.8.1 Class `bw.gate.NANDGate4`

Defined in bitwise/gate/NAND.py.

Four-input NAND gate.

11.8.2 __init__

```
__init__(
    input_1,
    input_2,
    input_3,
    input_4,
    output
)
```

Construct a new four-input NAND gate.

Args:

- `input_1`: An object of type `Wire`. The first input to the NAND gate.
- `input_2`: An object of type `Wire`. The second input to the NAND gate.
- `input_3`: An object of type `Wire`. The third input to the NAND gate.
- `input_4`: An object of type `Wire`. The fourth input to the NAND gate.
- `output`: An object of type `Wire`. The output of the NAND gate.

11.8.3 __str__

Print out the wire values of the NAND gate.

```
input_1: 0
input_2: 0
input_3: 0
input_4: 0
output: 0
```

11.8.4 __call__

```
__call__(
    input_1=None,
    input_2=None,
    input_3=None,
    input_4=None,
    output=None
)
```

Force specific values on the wires of the NAND gate.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

11.9 NORGate2

11.9.1 Class `bw.gate.NORGate2`

Defined in bitwise/gate/NOR.py.

Two-input NOR gate.

11.9.2 __init__

```
__init__(
    input_1,
    input_2,
    output
)
```

Construct a new two-input NOR gate.

Args:

- `input_1`: An object of type `Wire`. The first input to the NOR gate.
- `input_2`: An object of type `Wire`. The second input to the NOR gate.
- `output`: An object of type `Wire`. The output of the NOR gate.

11.9.3 __str__

Print out the wire values of the NOR gate.

```
input_1: 0
input_2: 0
output: 0
```

11.9.4 __call__

```
__call__(
    input_1=None,
    input_2=None,
    output=None
)
```

Force specific values on the wires of the NOR gate.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

11.10 NORGate3

11.10.1 Class `bw.gate.NORGate3`

Defined in bitwise/gate/NOR.py.

Three-input NOR gate.

11.10.2 __init__

```
__init__(
    input_1,
    input_2,
    input_3,
    output
)
```

Construct a new three-input NOR gate.

Args:

- `input_1`: An object of type `Wire`. The first input to the NOR gate.
- `input_2`: An object of type `Wire`. The second input to the NOR gate.
- `input_3`: An object of type `Wire`. The third input to the NOR gate.
- `output`: An object of type `Wire`. The output of the NOR gate.

11.10.3 __str__

Print out the wire values of the NOR gate.

```
input_1: 0
input_2: 0
input_3: 0
output: 0
```

11.10.4 __call__

```
__call__(
    input_1=None,
    input_2=None,
    input_3=None,
    output=None
)
```

Force specific values on the wires of the NOR gate.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

11.11 NORGate4

11.11.1 Class `bw.gate.NORGate4`

Defined in bitwise/gate/NOR.py.

Four-input NOR gate.

11.11.2 __init__

```
__init__(
    input_1,
    input_2,
    input_3,
    input_4,
    output
)
```

Construct a new four-input NOR gate.

Args:

- `input_1`: An object of type `Wire`. The first input to the NOR gate.
- `input_2`: An object of type `Wire`. The second input to the NOR gate.
- `input_3`: An object of type `Wire`. The third input to the NOR gate.
- `input_4`: An object of type `Wire`. The fourth input to the NOR gate.

- `output`: An object of type `Wire`. The output of the NOR gate.

11.11.3 __str__

Print out the wire values of the NOR gate.

```
input_1: 0
input_2: 0
input_3: 0
input_4: 0
output: 0
```

11.11.4 __call__

```
__call__(
    input_1=None,
    input_2=None,
    input_3=None,
    input_4=None,
    output=None
)
```

Force specific values on the wires of the NOR gate.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

11.12 NOTGate

11.12.1 Class `bw.gate.NOTGate`

Defined in bitwise/gate/NOT.py.

NOT gate.

11.12.2 __init__

```
__init__(
    input,
    output
)
```

Construct a new NOT gate.

Args:

- `input`: An object of type `Wire`. The input to the NOT gate.
- `output`: An object of type `Wire`. The output of the NOT gate.

11.12.3 __str__

Print out the wire values of the NOT gate.

```
input: 0
output: 0
```

11.12.4 __call__

```
__call__(
    input=None,
    output=None
)
```

Force specific values on the wires of the NOT gate.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

11.13 ORGate2

11.13.1 Class `bw.gate.ORGate2`

Defined in bitwise/gate/OR.py.

Two-input OR gate.

11.13.2 __init__

```
__init__(
    input_1,
    input_2,
    output
)
```

Construct a new two-input OR gate.

Args:

- `input_1`: An object of type `Wire`. The first input to the OR gate.
- `input_2`: An object of type `Wire`. The second input to the OR gate.
- `output`: An object of type `Wire`. The output of the OR gate.

11.13.3 __str__

Print out the wire values of the OR gate.

```
input_1: 0
input_2: 0
output: 0
```

11.13.4 __call__

```
__call__(
    input_1=None,
    input_2=None,
    output=None
)
```

Force specific values on the wires of the OR gate.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

11.14 ORGate3

11.14.1 Class `bw.gate.ORGate3`

Defined in bitwise/gate/OR.py.

Three-input OR gate.

11.14.2 __init__

```
__init__(
    input_1,
    input_2,
    input_3,
    output
)
```

Construct a new three-input OR gate.

Args:

- `input_1`: An object of type `Wire`. The first input to the OR gate.
- `input_2`: An object of type `Wire`. The second input to the OR gate.
- `input_3`: An object of type `Wire`. The third input to the OR gate.
- `output`: An object of type `Wire`. The output of the OR gate.

11.14.3 __str__

Print out the wire values of the OR gate.

```
input_1: 0
input_2: 0
input_3: 0
output: 0
```

11.14.4 __call__

```
__call__(
    input_1=None,
    input_2=None,
    input_3=None,
    output=None
)
```

Force specific values on the wires of the OR gate.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

11.15 ORGate4

11.15.1 Class `bw.gate.ORGate4`

Defined in bitwise/gate/OR.py.

Four-input OR gate.

11.15.2 __init__

```
__init__(
    input_1,
    input_2,
    input_3,
    input_4,
    output
)
```

Construct a new four-input OR gate.

Args:

- `input_1`: An object of type `Wire`. The first input to the OR gate.
- `input_2`: An object of type `Wire`. The second input to the OR gate.
- `input_3`: An object of type `Wire`. The third input to the OR gate.
- `input_4`: An object of type `Wire`. The fourth input to the OR gate.
- `output`: An object of type `Wire`. The output of the OR gate.

11.15.3 __str__

Print out the wire values of the OR gate.

```
input_1: 0
input_2: 0
input_3: 0
input_4: 0
output: 0
```

11.15.4 __call__

```
__call__(
    input_1=None,
    input_2=None,
    input_3=None,
    input_4=None,
    output=None
)
```

Force specific values on the wires of the OR gate.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

11.16 XNORGate2

11.16.1 Class `bw.gate.XNORGate2`

Defined in bitwise/gate/XNOR.py.

Two-input XNOR gate.

11.16.2 __init__

```
__init__(
    input_1,
    input_2,
    output
)
```

Construct a new two-input XNOR gate.

Args:

- `input_1`: An object of type `Wire`. The first input to the XNOR gate.
- `input_2`: An object of type `Wire`. The second input to the XNOR gate.
- `output`: An object of type `Wire`. The output of the XNOR gate.

11.16.3 __str__

Print out the wire values of the XNOR gate.

```
input_1: 0
input_2: 0
output: 0
```

11.16.4 __call__

```
__call__(
    input_1=None,
    input_2=None,
    output=None
)
```

Force specific values on the wires of the XNOR gate.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

11.17 XORGate2

11.17.1 Class `bw.gate.XORGate2`

Defined in bitwise/gate/XOR.py.

Two-input XOR gate.

11.17.2 __init__

```
__init__(
    input_1,
    input_2,
    output
)
```

Construct a new two-input XOR gate.

Args:

- `input_1`: An object of type `Wire`. The first input to the XOR gate.
- `input_2`: An object of type `Wire`. The second input to the XOR gate.
- `output`: An object of type `Wire`. The output of the XOR gate.

11.17.3 __str__

Print out the wire values of the XOR gate.

```
input_1: 0
input_2: 0
output: 0
```

11.17.4 __call__

```
__call__(
    input_1=None,
    input_2=None,
    output=None
)
```

Force specific values on the wires of the XOR gate.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

CHAPTER 12

Getting Started

12.1 Installation

Before installing, make sure your Python version is 3.5 or greater. Older versions may work properly but are unsupported. To check your Python version, use

```
$ python --version
```

in a terminal or command prompt.

If this requirement is not met, you can download the latest version of Python here.

Once Python is installed, you can install Bitwise using

```
$ pip install bitwise
```

If there are no errors, you can open a Python session and verify the installation by checking the version number:

```
In [1]: import bitwise as bw

In [2]: bw.__version__
Out[2]: '0.3'
```

Refer to the *changelog* for version details. In this documentation, it is canonical to have imported Bitwise as `bw`.

12.2 Basics

The use case of Bitwise is to model logic circuits in software; thus, it is useful to keep in mind that when you are using this library, you are actually *building a circuit* rather than writing a conventional software program. To facilitate this, there are a plethora of new features that are introduced.

12.2.1 Wires

By far the most important class defined by this library is `Wire`. As the name suggests, `Wire` is used to connect different logic elements together, just like a real physical wire. They can be instantiated like so:

```
In [1]: my_wire = bw.wire.Wire()

In [2]: my_other_wire = bw.wire.Wire()
```

This creates two objects of type `Wire`. Objects of this class have a single attribute, `value`, that can take on 1 of two binary values (0 or 1). The default `value` is 0, and it can be both accessed and mutated using `Wire.value`.

```
In [3]: my_wire.value
Out[3]: 0

In [4]: my_other_wire.value
Out[4]: 0

In [5]: my_wire.value = 1

In [6]: my_wire.value
Out[6]: 1

In [7]: my_wire.value = 0

In [8]: my_wire.value
Out[8]: 0
```

Other values will throw a `ValueError`.

Buses are used to group together sets of 4, 8, and 16 wires, as well as 7 wires, meant for use with the `SevenSegmentConverter` modules. They are implemented using the `Bus4`, `Bus8`, `Bus16`, and `BusSevenSegmentDisplay` classes, respectively, and can be instantiated by specifying the appropriate number of `Wire` objects

```
In [1]: my_bus = bw.wire.Bus4(my_wire_1, my_wire_2, my_wire_3, my_wire_4)
```

or with no arguments, which will automatically create wires with default `value` 0.

```
In [1]: my_bus = bw.wire.Bus4()
```

After instantiation, the `Wire` objects can be accessed using `Bus.wires`, and the wire values can be both accessed and mutated using `Bus.wire_values`.

12.2.2 Logic Elements

Alongside `Wire` and `Bus`, various logic elements are also at your disposal. These range from primitive logic gates to larger-scale logic circuits such as multiplexers and counters. In order to create new logic elements, usually a few `Wire` objects must be created first in order to make the necessary connections with the rest of your circuit. For example, the class that implements a two-input AND gate, `ANDGate2`, takes three arguments in its `__init__` method, all of type `Wire`:

```
In [1]: input_1 = bw.wire.Wire()

In [2]: input_2 = bw.wire.Wire()

```

(continues on next page)

(continued from previous page)

```
In [3]: output = bw.wire.Wire()

In [4]: my_AND_gate = bw.gate.ANDGate2(input_1, input_2, output)
```

Here, the value of `output` will take on the result of AND-ing the values of `input_1` and `input_2`. Note that all the arguments must be present and valid for proper instantiation.

Logic elements that require buses as arguments may then be instantiated. For example, the 4-bit parity generator, `ParityGenerator4`, receives two arguments: an object of type `Bus4` and an object of type `Wire`. It can be instantiated like so:

```
In [1]: my_bus = bw.wire.Bus4()

In [2]: output = bw.wire.Wire()

In [3]: my_parity_generator = bw.logic.ParityGenerator4(my_bus, output)
```

For a catalog of all logic elements available, refer to the *API documentation*.

12.2.3 Sensitivity

The concept of sensitivity is another key feature of this library. In a hardware circuit, when an input changes, the output changes immediately. In Bitwise, this type of behavior is accomplished by retaining a list of all the connections that an object of type `Wire` has, but it is akin to a sensitivity list in a true hardware description language like Verilog. To see sensitivity in action, consider again the `ANDGate2` example from the previous subsection:

```
In [1]: input_1 = bw.wire.Wire()

In [2]: input_2 = bw.wire.Wire()

In [3]: output = bw.wire.Wire()

In [4]: my_AND_gate = bw.gate.ANDGate2(input_1, input_2, output)
```

We can examine the value of `output` for every combination of values for `input_1` and `input_2`. Recall that the default value of a `Wire` object is 0.

```
In [5]: output.value
Out[5]: 0

In [6]: input_1.value = 1

In [7]: output.value
Out[7]: 0

In [8]: input_1.value = 0

In [9]: input_2.value = 1

In [10]: output.value
Out[10]: 0

In [11]: input_1.value = 1

In [12]: output.value
```

(continues on next page)

(continued from previous page)

```
Out[12]: 1

In [13]: input_1.value = 0

In [14]: input_2.value = 0

In [15]: output.value
Out[15]: 0
```

Notice that `output.value` reacts immediately to changes in the values of `input_1` and `input_2`, mimicking the behavior of a hardware circuit. (Moreover, we've verified that the two-input AND gate works as intended.)

12.2.4 Hierarchy

In order to build higher-level logic circuits, the concept of hierarchy must be introduced. Quite simply, logic elements can have instances of other logic elements, which in turn can have instances of yet other logic elements. The result is a hierarchical design pattern, with the primitive logic gates at the bottom (since they do not instantiate any other elements). For example, consider the following Python script, which defines a logic element with three inputs and one output. The value of `output` is the result of AND'ing the first two inputs and OR'ing the result with the third input.

```
import bitwise as bw

class MyLogicElement:
    def __init__(self, input_1, input_2, input_3, output):
        wire_1 = bw.wire.Wire()  # used as the output of the AND gate
        bw.gate.ANDGate2(input_1, input_2, wire_1)
        bw.gate.ORGate2(wire_1, input_3, output)
```

Notice, first and foremost, that the class has only one method, `__init__`, which only takes in arguments of type `Wire` (and bus types) and whose purpose is simply to make the necessary wire connections with the rest of the logic circuit. Notice also that the logic element itself instantiates an object of type `Wire`, `wire_1`. This is necessary in order to internally connect the output of the two-input AND gate to one of the inputs of the OR gate. Lastly, notice that both the inputs and the output of the logic element are given as arguments to the `__init__` method. Again, this is necessary so that both the inputs and the output are connected in some way to the rest of the logic circuit.

Objects of `MyLogicElement` can now be instantiated:

```
In [1]: input_1 = bw.wire.Wire()

In [2]: input_2 = bw.wire.Wire()

In [3]: input_3 = bw.wire.Wire()

In [4]: output = bw.wire.Wire()

In [5]: my_logic_element = MyLogicElement(input_1, input_2, input_3, output)
```

We can test various values for `input_1`, `input_2`, and `input_3` to verify that the circuit works as intended:

```
In [6]: output.value
Out[6]: 0

In [7]: input_1.value = 1

In [8]: output.value
```

(continues on next page)

(continued from previous page)

```
Out[8]: 0

In [9]: input_2.value = 1

In [10]: output.value
Out[10]: 1

In [11]: input_1.value = 0

In [12]: input_2.value = 0

In [13]: output.value
Out[13]: 0

In [14]: input_3.value = 1

In [15]: output.value
Out[15]: 1
```

Additionally, objects of `MyLogicElement` can now be instantiated in other logic elements and circuits.

12.2.5 Example

As a short example, let us construct a 2-bit adder. An adder simply takes two inputs, of a certain width, and outputs the sum. In this case, since we have two inputs of width 2, four inputs to the adder are needed. Additionally, three outputs are needed, since the sum of two 2-bit numbers can be at most 3 bits wide.

Before a full 2-bit adder can be constructed, we first need a 1-bit adder. This adder must have not two, but three inputs, since we need one input to be a "carry-in" input from the previous 1-bit adder. Two outputs are also needed. Skipping a few details, the following script defines a class that simulates our 1-bit adder:

```
import bitwise as bw

class OneBitAdder:
    def __init__(self, carry_in, input_1, input_2, sum_1, sum_2):
        # these wires connect the appropriate gates together (trust me, it works)
        wire_1 = bw.wireWire()
        wire_2 = bw.wire.Wire()
        wire_3 = bw.wire.Wire()

        bw.gate.XORGate2(input_1, input_2, wire_1)
        bw.gate.XORGate2(carry_in, wire_1, sum_2)
        bw.gate.ANDGate2(input_1, input_2, wire_2)
        bw.gate.ANDGate2(carry_in, wire_1, wire_3)
        bw.gate.ORGate2(wire_2, wire_3, sum_1)
```

Here, `sum_1` and `sum_2` are the most and least significant bits of the sum, respectively. It can be verified that this element behaves as intended:

```
In [1]: carry_in = bw.wire.Wire()

In [2]: input_1 = bw.wire.Wire()

In [3]: input_2 = bw.wire.Wire()
```

(continues on next page)

(continued from previous page)

```
In [4]: sum_1 = bw.wire.Wire()

In [5]: sum_2 = bw.wire.Wire()

In [6]: myOneBitAdder = OneBitAdder(carry_in, input_1, input_2, sum_1, sum_2)

In [7]: sum_1.value
Out[7]: 0

In [8]: sum_2.value
Out[8]: 0

In [9]: input_1.value = 1

In [10]: sum_1.value
Out[10]: 0

In [11]: sum_2.value
Out[11]: 1

In [12]: input_2.value = 1

In [13]: sum_1.value
Out[13]: 1

In [14]: sum_2.value
Out[14]: 0

In [15]: carry_in.value = 1

In [16]: sum_1.value
Out[16]: 1

In [17]: sum_2.value
Out[17]: 1
```

A 2-bit adder may now be constructed by creating two instances of `OneBitAdder` in a hierarchical design pattern and connecting the wires appropriately:

```
class TwoBitAdder:
    def __init__(self, input_1_a, input_1_b, input_2_a, input_2_b, sum_1, sum_2, sum_
↪3):
        wire_1 = bw.wire.Wire()  # used to connect the two 1-bit adders
        gnd = bw.wire.Wire()
        gnd.value = 0

        OneBitAdder(gnd, input_1_b, input_2_b, wire_1, sum_3)
        OneBitAdder(wire_1, input_1_a, input_2_a, sum_1, sum_2)
```

Here, `input_1_a` and `input_1_b` are the most and least significant bits of the first input, respectively, `input_2_a` and `input_2_b` are the most and least significant bits of the second input, respectively, and `sum_1` and `sum_3` are the most and least significant bits of the sum, respectively. Since the least significant adder has no carry-in, the `gnd` wire is used for the `carry_in` input. The `wire_1` wire is used to connect the most significant bit of the sum from the least significant adder with the carry-in of the most significant adder.

Again, it can be verified that this element behaves as intended by trying out a few test cases:

```
In [1]: i_1_a = bw.wire.Wire()

In [2]: i_1_b = bw.wire.Wire()

In [3]: i_2_a = bw.wire.Wire()

In [4]: i_2_b = bw.wire.Wire()

In [5]: sum_1 = bw.wire.Wire()

In [6]: sum_2 = bw.wire.Wire()

In [7]: sum_3 = bw.wire.Wire()

In [8]: myTwoBitAdder = TwoBitAdder(i_1_a, i_1_b, i_2_a, i_2_b, sum_1, sum_2, sum_3)

In [9]: sum_1.value
Out[9]: 0

In [10]: sum_2.value
Out[10]: 0

In [11]: sum_3.value
Out[11]: 0

In [12]: i_1_b.value = 1

In [13]: i_2_b.value = 1

In [14]: sum_1.value
Out[14]: 0

In [15]: sum_2.value
Out[15]: 1

In [16]: sum_3.value
Out[16]: 0

In [17]: i_1_a.value = 1

In [18]: sum_1.value
Out[18]: 1

In [19]: sum_2.value
Out[19]: 0

In [20]: sum_3.value
Out[20]: 0

In [21]: i_2_a.value = 1

In [22]: sum_1.value
Out[22]: 1

In [23]: sum_2.value
Out[23]: 1
```

(continues on next page)

(continued from previous page)

```
In [24]: sum_3.value
Out[24]: 0
```

All of the sums are as expected.

12.3 Issues

Please post all bugs, issues, and feature requests in the issues section of the Github repository.

CHAPTER 13

Logic

13.1 BitwiseAND4

13.1.1 Class `bw.logic.BitwiseAND4`

Defined in bitwise/logic/AND.py.

4-bit bitwise AND circuit.

13.1.2 __init__

```
__init__(
    a_bus,
    b_bus,
    output_bus
)
```

Construct a new 4-bit bitwise AND circuit.

Args:

- `a_bus`: An object of type `Bus4`. The first input.
- `b_bus`: An object of type `Bus4`. The second input.
- `output_bus`: An object of type `Bus4`. The output of the bitwise AND operation.

Raises:

- `TypeError`: If either `a_bus`, `b_bus`, or `output_bus` is not a bus of width 4.

13.1.3 __str__

Print out the wire values of the 4-bit bitwise AND circuit.

```
a_bus: (0, 0, 0, 0)
b_bus: (0, 0, 0, 0)
output_bus: (0, 0, 0, 0)
```

13.1.4 __call__

```
__call__(
    a_bus=None,
    b_bus=None,
    output_bus=None
)
```

Force specific values on the wires of the 4-bit bitwise AND circuit.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

13.2 BitwiseAND8

13.2.1 Class `bw.logic.BitwiseAND8`

Defined in bitwise/logic/AND.py.

8-bit bitwise AND circuit.

13.2.2 __init__

```
__init__(
    a_bus,
    b_bus,
    output_bus
)
```

Construct a new 8-bit bitwise AND circuit.

Args:

- `a_bus`: An object of type `Bus8`. The first input.
- `b_bus`: An object of type `Bus8`. The second input.
- `output_bus`: An object of type `Bus8`. The output of the bitwise AND operation.

Raises:

- `TypeError`: If either `a_bus`, `b_bus`, or `output_bus` is not a bus of width 8.

13.2.3 __str__

Print out the wire values of the 8-bit bitwise AND circuit.

```
a_bus: (0, 0, 0, 0, 0, 0, 0, 0)
b_bus: (0, 0, 0, 0, 0, 0, 0, 0)
output_bus: (0, 0, 0, 0, 0, 0, 0, 0)
```

13.2.4 __call__

```
__call__(
    a_bus=None,
    b_bus=None,
    output_bus=None
)
```

Force specific values on the wires of the 8-bit bitwise AND circuit.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

13.3 BitwiseAND16

13.3.1 Class `bw.logic.BitwiseAND16`

Defined in bitwise/logic/AND.py.

16-bit bitwise AND circuit.

13.3.2 __init__

```
__init__(
    a_bus,
    b_bus,
    output_bus
)
```

Construct a new 16-bit bitwise AND circuit.

Args:

- `a_bus`: An object of type `Bus16`. The first input.
- `b_bus`: An object of type `Bus16`. The second input.
- `output_bus`: An object of type `Bus16`. The output of the bitwise AND operation.

Raises:

- `TypeError`: If either `a_bus`, `b_bus`, or `output_bus` is not a bus of width 16.

13.3.3 __str__

Print out the wire values of the 16-bit bitwise AND circuit.

```
a_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
b_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
output_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
```

13.3.4 __call__

```
__call__(
    a_bus=None,
    b_bus=None,
    output_bus=None
)
```

Force specific values on the wires of the 16-bit bitwise AND circuit.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

13.4 BitwiseNAND4

13.4.1 Class `bw.logic.BitwiseNAND4`

Defined in bitwise/logic/NAND.py.

4-bit bitwise NAND circuit.

13.4.2 __init__

```
__init__(
    a_bus,
    b_bus,
    output_bus
)
```

Construct a new 4-bit bitwise NAND circuit.

Args:

- `a_bus`: An object of type `Bus4`. The first input.
- `b_bus`: An object of type `Bus4`. The second input.
- `output_bus`: An object of type `Bus4`. The output of the bitwise NAND operation.

Raises:

- `TypeError`: If either `a_bus`, `b_bus`, or `output_bus` is not a bus of width 4.

13.4.3 __str__

Print out the wire values of the 4-bit bitwise NAND circuit.

```
a_bus: (0, 0, 0, 0)
b_bus: (0, 0, 0, 0)
output_bus: (0, 0, 0, 0)
```

13.4.4 __call__

```
__call__(
    a_bus=None,
    b_bus=None,
    output_bus=None
)
```

Force specific values on the wires of the 4-bit bitwise NAND circuit.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

13.5 BitwiseNAND8

13.5.1 Class `bw.logic.BitwiseNAND8`

Defined in bitwise/logic/NAND.py.

8-bit bitwise NAND circuit.

13.5.2 __init__

```
__init__(
    a_bus,
    b_bus,
    output_bus
)
```

Construct a new 8-bit bitwise NAND circuit.

Args:

- `a_bus`: An object of type `Bus8`. The first input.
- `b_bus`: An object of type `Bus8`. The second input.
- `output_bus`: An object of type `Bus8`. The output of the bitwise NAND operation.

Raises:

- `TypeError`: If either `a_bus`, `b_bus`, or `output_bus` is not a bus of width 8.

13.5.3 __str__

Print out the wire values of the 8-bit bitwise NAND circuit.

```
a_bus: (0, 0, 0, 0, 0, 0, 0, 0)
b_bus: (0, 0, 0, 0, 0, 0, 0, 0)
output_bus: (0, 0, 0, 0, 0, 0, 0, 0)
```

13.5.4 __call__

```
__call__(
    a_bus=None,
    b_bus=None,
    output_bus=None
)
```

Force specific values on the wires of the 8-bit bitwise NAND circuit.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

13.6 BitwiseNAND16

13.6.1 Class `bw.logic.BitwiseNAND16`

Defined in bitwise/logic/NAND.py.

16-bit bitwise NAND circuit.

13.6.2 __init__

```
__init__(
    a_bus,
    b_bus,
    output_bus
)
```

Construct a new 16-bit bitwise NAND circuit.

Args:

- `a_bus`: An object of type `Bus16`. The first input.
- `b_bus`: An object of type `Bus16`. The second input.
- `output_bus`: An object of type `Bus16`. The output of the bitwise NAND operation.

Raises:

- `TypeError`: If either `a_bus`, `b_bus`, or `output_bus` is not a bus of width 16.

13.6.3 __str__

Print out the wire values of the 16-bit bitwise NAND circuit.

```
a_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
b_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
output_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
```

13.6.4 __call__

```
__call__(
    a_bus=None,
    b_bus=None,
    output_bus=None
)
```

Force specific values on the wires of the 16-bit bitwise NAND circuit.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

13.7 BitwiseNOR4

13.7.1 Class `bw.logic.BitwiseNOR4`

Defined in bitwise/logic/NOR.py.

4-bit bitwise NOR circuit.

13.7.2 __init__

```
__init__(
    a_bus,
    b_bus,
    output_bus
)
```

Construct a new 4-bit bitwise NOR circuit.

Args:

- `a_bus`: An object of type `Bus4`. The first input.
- `b_bus`: An object of type `Bus4`. The second input.
- `output_bus`: An object of type `Bus4`. The output of the bitwise NOR operation.

Raises:

- `TypeError`: If either `a_bus`, `b_bus`, or `output_bus` is not a bus of width 4.

13.7.3 __str__

Print out the wire values of the 4-bit bitwise NOR circuit.

```
a_bus: (0, 0, 0, 0)
b_bus: (0, 0, 0, 0)
output_bus: (0, 0, 0, 0)
```

13.7.4 __call__

```
__call__(
    a_bus=None,
    b_bus=None,
    output_bus=None
)
```

Force specific values on the wires of the 4-bit bitwise NOR circuit.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

13.8 BitwiseNOR8

13.8.1 Class `bw.logic.BitwiseNOR8`

Defined in bitwise/logic/NOR.py.

8-bit bitwise NOR circuit.

13.8.2 __init__

```
__init__(
    a_bus,
    b_bus,
    output_bus
)
```

Construct a new 8-bit bitwise NOR circuit.

Args:

- `a_bus`: An object of type `Bus8`. The first input.
- `b_bus`: An object of type `Bus8`. The second input.
- `output_bus`: An object of type `Bus8`. The output of the bitwise NOR operation.

Raises:

- `TypeError`: If either `a_bus`, `b_bus`, or `output_bus` is not a bus of width 8.

13.8.3 __str__

Print out the wire values of the 8-bit bitwise NOR circuit.

```
a_bus: (0, 0, 0, 0, 0, 0, 0, 0)
b_bus: (0, 0, 0, 0, 0, 0, 0, 0)
output_bus: (0, 0, 0, 0, 0, 0, 0, 0)
```

13.8.4 __call__

```
__call__(
    a_bus=None,
    b_bus=None,
    output_bus=None
)
```

Force specific values on the wires of the 8-bit bitwise NOR circuit.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

13.9 BitwiseNOR16

13.9.1 Class `bw.logic.BitwiseNOR16`

Defined in bitwise/logic/NOR.py.

16-bit bitwise NOR circuit.

13.9.2 __init__

```
__init__(
    a_bus,
    b_bus,
    output_bus
)
```

Construct a new 16-bit bitwise NOR circuit.

Args:

- `a_bus`: An object of type `Bus16`. The first input.
- `b_bus`: An object of type `Bus16`. The second input.
- `output_bus`: An object of type `Bus16`. The output of the bitwise NOR operation.

Raises:

- `TypeError`: If either `a_bus`, `b_bus`, or `output_bus` is not a bus of width 16.

13.9.3 __str__

Print out the wire values of the 16-bit bitwise NOR circuit.

```
a_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
b_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
output_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
```

13.9.4 __call__

```
__call__(
    a_bus=None,
    b_bus=None,
    output_bus=None
)
```

Force specific values on the wires of the 16-bit bitwise NOR circuit.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

13.10 BitwiseNOT4

13.10.1 Class `bw.logic.BitwiseNOT4`

Defined in bitwise/logic/NOT.py.

4-bit bitwise NOT circuit.

13.10.2 __init__

```
__init__(
    input_bus,
    output_bus
)
```

Construct a new 4-bit bitwise NOT circuit.

Args:

- `input_bus`: An object of type `Bus4`. The input to the bitwise NOT operation.
- `output_bus`: An object of type `Bus4`. The output of the bitwise NOT operation.

Raises:

- `TypeError`: If either `input_bus` or `output_bus` is not a bus of width 4.

13.10.3 __str__

Print out the wire values of the 4-bit bitwise NOT circuit.

```
input_bus: (0, 0, 0, 0)
output_bus: (0, 0, 0, 0)
```

13.10.4 __call__

```
__call__(
    input_bus=None,
    output_bus=None
)
```

Force specific values on the wires of the 4-bit bitwise NOT circuit.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

13.11 BitwiseNOT8

13.11.1 Class `bw.logic.BitwiseNOT8`

Defined in bitwise/logic/NOT.py.

8-bit bitwise NOT circuit.

13.11.2 __init__

```
__init__(
    input_bus,
    output_bus
)
```

Construct a new 8-bit bitwise NOT circuit.

Args:

- `input_bus`: An object of type `Bus8`. The input to the bitwise NOT operation.
- `output_bus`: An object of type `Bus8`. The output of the bitwise NOT operation.

Raises:

- `TypeError`: If either `input_bus` or `output_bus` is not a bus of width 8.

13.11.3 __str__

Print out the wire values of the 8-bit bitwise NOT circuit.

```
input_bus: (0, 0, 0, 0, 0, 0, 0, 0)
output_bus: (0, 0, 0, 0, 0, 0, 0, 0)
```

13.11.4 __call__

```
__call__(
    input_bus=None,
    output_bus=None
)
```

Force specific values on the wires of the 8-bit bitwise NOT circuit.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

13.12 BitwiseNOT16

13.12.1 Class `bw.logic.BitwiseNOT16`

Defined in bitwise/logic/NOT.py.

16-bit bitwise NOT circuit.

13.12.2 __init__

```
__init__(
    input_bus,
    output_bus
)
```

Construct a new 16-bit bitwise NOT circuit.

Args:

- `input_bus`: An object of type `Bus16`. The input to the bitwise NOT operation.
- `output_bus`: An object of type `Bus16`. The output of the bitwise NOT operation.

Raises:

- `TypeError`: If either `input_bus` or `output_bus` is not a bus of width 16.

13.12.3 __str__

Print out the wire values of the 16-bit bitwise NOT circuit.

```
input_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
output_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
```

13.12.4 __call__

```
__call__(
    input_bus=None,
    output_bus=None
)
```

Force specific values on the wires of the 16-bit bitwise NOT circuit.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

13.13 BitwiseOR4

13.13.1 Class `bw.logic.BitwiseOR4`

Defined in bitwise/logic/OR.py.

4-bit bitwise OR circuit.

13.13.2 __init__

```
__init__(
    a_bus,
    b_bus,
    output_bus
)
```

Construct a new 4-bit bitwise OR circuit.

Args:

- `a_bus`: An object of type `Bus4`. The first input.
- `b_bus`: An object of type `Bus4`. The second input.
- `output_bus`: An object of type `Bus4`. The output of the bitwise OR operation.

Raises:

- `TypeError`: If either `a_bus`, `b_bus`, or `output_bus` is not a bus of width 4.

13.13.3 __str__

Print out the wire values of the 4-bit bitwise OR circuit.

```
a_bus: (0, 0, 0, 0)
b_bus: (0, 0, 0, 0)
output_bus: (0, 0, 0, 0)
```

13.13.4 __call__

```
__call__(
    a_bus=None,
    b_bus=None,
    output_bus=None
)
```

Force specific values on the wires of the 4-bit bitwise OR circuit.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

13.14 BitwiseOR8

13.14.1 Class `bw.logic.BitwiseOR8`

Defined in bitwise/logic/OR.py.

8-bit bitwise OR circuit.

13.14.2 __init__

```
__init__(
    a_bus,
    b_bus,
    output_bus
)
```

Construct a new 8-bit bitwise OR circuit.

Args:

- `a_bus`: An object of type `Bus8`. The first input.
- `b_bus`: An object of type `Bus8`. The second input.
- `output_bus`: An object of type `Bus8`. The output of the bitwise OR operation.

Raises:

- `TypeError`: If either `a_bus`, `b_bus`, or `output_bus` is not a bus of width 8.

13.14.3 __str__

Print out the wire values of the 8-bit bitwise OR circuit.

```
a_bus: (0, 0, 0, 0, 0, 0, 0, 0)
b_bus: (0, 0, 0, 0, 0, 0, 0, 0)
output_bus: (0, 0, 0, 0, 0, 0, 0, 0)
```

13.14.4 __call__

```
__call__(
    a_bus=None,
    b_bus=None,
    output_bus=None
)
```

Force specific values on the wires of the 8-bit bitwise OR circuit.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

13.15 BitwiseOR16

13.15.1 Class `bw.logic.BitwiseOR16`

Defined in bitwise/logic/OR.py.

16-bit bitwise OR circuit.

13.15.2 __init__

```
__init__(
    a_bus,
    b_bus,
    output_bus
)
```

Construct a new 16-bit bitwise OR circuit.

Args:

- `a_bus`: An object of type `Bus16`. The first input.
- `b_bus`: An object of type `Bus16`. The second input.
- `output_bus`: An object of type `Bus16`. The output of the bitwise OR operation.

Raises:

- `TypeError`: If either `a_bus`, `b_bus`, or `output_bus` is not a bus of width 16.

13.15.3 __str__

Print out the wire values of the 16-bit bitwise OR circuit.

```
a_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
b_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
output_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
```

13.15.4 __call__

```
__call__(
    a_bus=None,
    b_bus=None,
    output_bus=None
)
```

Force specific values on the wires of the 16-bit bitwise OR circuit.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

13.16 BitwiseXNOR4

13.16.1 Class `bw.logic.BitwiseXNOR4`

Defined in bitwise/logic/XNOR.py.

4-bit bitwise XNOR circuit.

13.16.2 __init__

```
__init__(
    a_bus,
    b_bus,
    output_bus
)
```

Construct a new 4-bit bitwise XNOR circuit.

Args:

- `a_bus`: An object of type `Bus4`. The first input.
- `b_bus`: An object of type `Bus4`. The second input.
- `output_bus`: An object of type `Bus4`. The output of the bitwise XNOR operation.

Raises:

- `TypeError`: If either `a_bus`, `b_bus`, or `output_bus` is not a bus of width 4.

13.16.3 __str__

Print out the wire values of the 4-bit bitwise XNOR circuit.

```
a_bus: (0, 0, 0, 0)
b_bus: (0, 0, 0, 0)
output_bus: (0, 0, 0, 0)
```

13.16.4 __call__

```
__call__(
    a_bus=None,
    b_bus=None,
    output_bus=None
)
```

Force specific values on the wires of the 4-bit bitwise XNOR circuit.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

13.17 BitwiseXNOR8

13.17.1 Class `bw.logic.BitwiseXNOR8`

Defined in bitwise/logic/XNOR.py.

8-bit bitwise XNOR circuit.

13.17.2 __init__

```
__init__(
    a_bus,
    b_bus,
    output_bus
)
```

Construct a new 8-bit bitwise XNOR circuit.

Args:

- `a_bus`: An object of type `Bus8`. The first input.
- `b_bus`: An object of type `Bus8`. The second input.
- `output_bus`: An object of type `Bus8`. The output of the bitwise XNOR operation.

Raises:

- `TypeError`: If either `a_bus`, `b_bus`, or `output_bus` is not a bus of width 8.

13.17.3 __str__

Print out the wire values of the 8-bit bitwise XNOR circuit.

```
a_bus: (0, 0, 0, 0, 0, 0, 0, 0)
b_bus: (0, 0, 0, 0, 0, 0, 0, 0)
output_bus: (0, 0, 0, 0, 0, 0, 0, 0)
```

13.17.4 __call__

```
__call__(
    a_bus=None,
    b_bus=None,
    output_bus=None
)
```

Force specific values on the wires of the 8-bit bitwise XNOR circuit.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

13.18 BitwiseXNOR16

13.18.1 Class `bw.logic.BitwiseXNOR16`

Defined in bitwise/logic/XNOR.py.

16-bit bitwise XNOR circuit.

13.18.2 __init__

```
__init__(
    a_bus,
    b_bus,
    output_bus
)
```

Construct a new 16-bit bitwise XNOR circuit.

Args:

- `a_bus`: An object of type `Bus16`. The first input.
- `b_bus`: An object of type `Bus16`. The second input.
- `output_bus`: An object of type `Bus16`. The output of the bitwise XNOR operation.

Raises:

- `TypeError`: If either `a_bus`, `b_bus`, or `output_bus` is not a bus of width 16.

13.18.3 __str__

Print out the wire values of the 16-bit bitwise XNOR circuit.

```
a_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
b_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
output_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
```

13.18.4 __call__

```
__call__(
    a_bus=None,
    b_bus=None,
    output_bus=None
)
```

Force specific values on the wires of the 16-bit bitwise XNOR circuit.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

13.19 BitwiseXOR4

13.19.1 Class `bw.logic.BitwiseXOR4`

Defined in bitwise/logic/XOR.py.

4-bit bitwise XOR circuit.

13.19.2 __init__

```
__init__(
    a_bus,
    b_bus,
    output_bus
)
```

Construct a new 4-bit bitwise XOR circuit.

Args:

- `a_bus`: An object of type `Bus4`. The first input.
- `b_bus`: An object of type `Bus4`. The second input.
- `output_bus`: An object of type `Bus4`. The output of the bitwise XOR operation.

Raises:

- `TypeError`: If either `a_bus`, `b_bus`, or `output_bus` is not a bus of width 4.

13.19.3 __str__

Print out the wire values of the 4-bit bitwise XOR circuit.

```
a_bus: (0, 0, 0, 0)
b_bus: (0, 0, 0, 0)
output_bus: (0, 0, 0, 0)
```

13.19.4 __call__

```
__call__(
    a_bus=None,
    b_bus=None,
    output_bus=None
)
```

Force specific values on the wires of the 4-bit bitwise XOR circuit.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

13.20 BitwiseXOR8

13.20.1 Class `bw.logic.BitwiseXOR8`

Defined in bitwise/logic/XOR.py.

8-bit bitwise XOR circuit.

13.20.2 __init__

```
__init__(
    a_bus,
    b_bus,
    output_bus
)
```

Construct a new 8-bit bitwise XOR circuit.

Args:

- `a_bus`: An object of type `Bus8`. The first input.
- `b_bus`: An object of type `Bus8`. The second input.
- `output_bus`: An object of type `Bus8`. The output of the bitwise XOR operation.

Raises:

- `TypeError`: If either `a_bus`, `b_bus`, or `output_bus` is not a bus of width 8.

13.20.3 __str__

Print out the wire values of the 8-bit bitwise XOR circuit.

```
a_bus: (0, 0, 0, 0, 0, 0, 0, 0)
b_bus: (0, 0, 0, 0, 0, 0, 0, 0)
output_bus: (0, 0, 0, 0, 0, 0, 0, 0)
```

13.20.4 __call__

```
__call__(
    a_bus=None,
    b_bus=None,
    output_bus=None
)
```

Force specific values on the wires of the 8-bit bitwise XOR circuit.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

13.21 BitwiseXOR16

13.21.1 Class `bw.logic.BitwiseXOR16`

Defined in bitwise/logic/XOR.py.

16-bit bitwise XOR circuit.

13.21.2 __init__

```
__init__(
    a_bus,
    b_bus,
    output_bus
)
```

Construct a new 16-bit bitwise XOR circuit.

Args:

- `a_bus`: An object of type `Bus16`. The first input.
- `b_bus`: An object of type `Bus16`. The second input.
- `output_bus`: An object of type `Bus16`. The output of the bitwise XOR operation.

Raises:

- `TypeError`: If either `a_bus`, `b_bus`, or `output_bus` is not a bus of width 16.

13.21.3 __str__

Print out the wire values of the 16-bit bitwise XOR circuit.

```
a_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
b_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
output_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
```

13.21.4 __call__

```
__call__(
    a_bus=None,
    b_bus=None,
    output_bus=None
)
```

Force specific values on the wires of the 16-bit bitwise XOR circuit.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

13.22 Comparator3

13.22.1 Class `bw.logic.Comparator3`

Defined in bitwise/logic/COMP.py.

3-bit logical comparator.

13.22.2 __init__

```
__init__(
    a_bus,
    b_bus,
    greater_than,
    equal_to,
    less_than
)
```

Construct a new 3-bit logical comparator.

Args:

- `a_bus`: An object of type `Bus4`. The number to be compared. `a_bus[1]` and `a_bus[3]` are the most and least significant bit, respectively. `a_bus[0]` is the sign bit.
- `b_bus`: An object of type `Bus4`. The number to be compared against. `b_bus[1]` and `b_bus[3]` are the most and least significant bit, respectively. `b_bus[0]` is the sign bit.
- `greater_than`: An object of type `Wire`. The greater-than indicator.
- `equal_to`: An object of type `Wire`. The equal-to indicator.

- `less_than`: An object of type `Wire`. The less-than indicator.

Raises:

- `TypeError`: If either `a_bus` or `b_bus` is not a bus of width 4.

13.22.3 __str__

Print out the wire values of the 3-bit logical comparator.

```
a_bus: (0, 0, 0, 0)
b_bus: (0, 0, 0, 0)
greater_than: 0
equal_to: 0
less_than: 0
```

13.22.4 __call__

```
__call__(
    a_bus=None,
    b_bus=None,
    greater_than=None,
    equal_to=None,
    less_than=None
)
```

Force specific values on the wires of the 3-bit logical comparator.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

13.23 Comparator7

13.23.1 Class `bw.logic.Comparator7`

Defined in bitwise/logic/COMP.py.

7-bit logical comparator.

13.23.2 __init__

```
__init__(
    a_bus,
    b_bus,
    greater_than,
    equal_to,
    less_than
)
```

Construct a new 7-bit logical comparator.

Args:

- `a_bus`: An object of type `Bus8`. The number to be compared. `a_bus[1]` and `a_bus[7]` are the most and least significant bit, respectively. `a_bus[0]` is the sign bit.
- `b_bus`: An object of type `Bus8`. The number to be compared against. `b_bus[1]` and `b_bus[7]` are the most and least significant bit, respectively. `b_bus[0]` is the sign bit.
- `greater_than`: An object of type `Wire`. The greater-than indicator.
- `equal_to`: An object of type `Wire`. The equal-to indicator.
- `less_than`: An object of type `Wire`. The less-than indicator.

Raises:

- `TypeError`: If either `a_bus` or `b_bus` is not a bus of width 8.

13.23.3 __str__

Print out the wire values of the 7-bit logical comparator.

```
a_bus: (0, 0, 0, 0, 0, 0, 0, 0)
b_bus: (0, 0, 0, 0, 0, 0, 0, 0)
greater_than: 0
equal_to: 0
less_than: 0
```

13.23.4 __call__

```
__call__(
    a_bus=None,
    b_bus=None,
    greater_than=None,
    equal_to=None,
    less_than=None
)
```

Force specific values on the wires of the 7-bit logical comparator.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

13.24 Comparator15

13.24.1 Class `bw.logic.Comparator15`

Defined in bitwise/logic/COMP.py.

15-bit logical comparator.

13.24.2 __init__

```
__init__(
    a_bus,
    b_bus,
    greater_than,
    equal_to,
    less_than
)
```

Construct a new 15-bit logical comparator.

Args:

- `a_bus`: An object of type `Bus16`. The number to be compared. `a_bus[1]` and `a_bus[15]` are the most and least significant bit, respectively. `a_bus[0]` is the sign bit.
- `b_bus`: An object of type `Bus16`. The number to be compared against. `b_bus[1]` and `b_bus[15]` are the most and least significant bit, respectively. `b_bus[0]` is the sign bit.
- `greater_than`: An object of type `Wire`. The greater-than indicator.
- `equal_to`: An object of type `Wire`. The equal-to indicator.
- `less_than`: An object of type `Wire`. The less-than indicator.

Raises:

- `TypeError`: If either `a_bus` or `b_bus` is not a bus of width 16.

13.24.3 __str__

Print out the wire values of the 15-bit logical comparator.

```
a_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
b_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
greater_than: 0
equal_to: 0
less_than: 0
```

13.24.4 __call__

```
__call__(
    a_bus=None,
    b_bus=None,
    greater_than=None,
    equal_to=None,
    less_than=None
)
```

Force specific values on the wires of the 15-bit logical comparator.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

13.25 ParityChecker4

13.25.1 Class `bw.logic.ParityChecker4`

Defined in bitwise/logic/PAR.py.

4-bit even parity checker.

13.25.2 __init__

```
__init__(
    input_bus,
    parity_bit,
    error
)
```

Construct a new 4-bit even parity checker.

Args:

- `input_bus`: An object of type `Bus4`. The input to the parity checker.
- `parity_bit`: An object of type `Wire`. The parity bit.
- `error`: An object of type `Wire`. The error indicator.

Raises:

- `TypeError`: If `input_bus` is not a bus of width 4.

13.25.3 __str__

Print out the wire values of the 4-bit even parity checker.

```
input_bus: (0, 0, 0, 0)
parity_bit: 0
error: 0
```

13.25.4 __call__

```
__call__(
    input_bus=None,
    parity_bit=None,
    error=None
)
```

Force specific values on the wires of the 4-bit even parity checker.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

13.26 ParityChecker8

13.26.1 Class `bw.logic.ParityChecker8`

Defined in bitwise/logic/PAR.py.

8-bit even parity checker.

13.26.2 __init__

```
__init__(
    input_bus,
    parity_bit,
    error
)
```

Construct a new 8-bit even parity checker.

Args:

- `input_bus`: An object of type `Bus8`. The input to the parity checker.
- `parity_bit`: An object of type `Wire`. The parity bit.
- `error`: An object of type `Wire`. The error indicator.

Raises:

- `TypeError`: If `input_bus` is not a bus of width 8.

13.26.3 __str__

Print out the wire values of the 8-bit even parity checker.

```
input_bus: (0, 0, 0, 0, 0, 0, 0, 0)
parity_bit: 0
error: 0
```

13.26.4 __call__

```
__call__(
    input_bus=None,
    parity_bit=None,
    error=None
)
```

Force specific values on the wires of the 8-bit even parity checker.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

13.27 ParityChecker16

13.27.1 Class `bw.logic.ParityChecker16`

Defined in bitwise/logic/PAR.py.

16-bit even parity checker.

13.27.2 __init__

```
__init__(
    input_bus,
    parity_bit,
    error
)
```

Construct a new 16-bit even parity checker.

Args:

- `input_bus`: An object of type `Bus16`. The input to the parity checker.
- `parity_bit`: An object of type `Wire`. The parity bit.
- `error`: An object of type `Wire`. The error indicator.

Raises:

- `TypeError`: If `input_bus` is not a bus of width 16.

13.27.3 __str__

Print out the wire values of the 16-bit even parity checker.

```
input_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
parity_bit: 0
error: 0
```

13.27.4 __call__

```
__call__(
    input_bus=None,
    parity_bit=None,
    error=None
)
```

Force specific values on the wires of the 16-bit even parity checker.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

13.28 ParityGenerator4

13.28.1 Class `bw.logic.ParityGenerator4`

Defined in bitwise/logic/PAR.py.

4-bit even parity generator.

13.28.2 __init__

```
__init__(
    input_bus,
    parity_bit
)
```

Construct a new 4-bit even parity generator.

Args:

- `input_bus`: An object of type `Bus4`. The input to the parity generator.
- `parity_bit`: An object of type `Wire`. The parity bit.

Raises:

- `TypeError`: If `input_bus` is not a bus of width 4.

13.28.3 __str__

Print out the wire values of the 4-bit even parity generator.

```
input_bus: (0, 0, 0, 0)
parity_bit: 0
```

13.28.4 __call__

```
__call__(
    input_bus=None,
    parity_bit=None
)
```

Force specific values on the wires of the 4-bit even parity generator.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

13.29 ParityGenerator8

13.29.1 Class `bw.logic.ParityGenerator8`

Defined in bitwise/logic/PAR.py.

8-bit even parity generator.

13.29.2 __init__

```
__init__(
    input_bus,
    parity_bit
)
```

Construct a new 8-bit even parity generator.

Args:

- `input_bus`: An object of type `Bus8`. The input to the parity generator.
- `parity_bit`: An object of type `Wire`. The parity bit.

Raises:

- `TypeError`: If `input_bus` is not a bus of width 8.

13.29.3 __str__

Print out the wire values of the 8-bit even parity generator.

```
input_bus: (0, 0, 0, 0, 0, 0, 0, 0)
parity_bit: 0
```

13.29.4 __call__

```
__call__(
    input_bus=None,
    parity_bit=None
)
```

Force specific values on the wires of the 8-bit even parity generator.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

13.30 ParityGenerator16

13.30.1 Class `bw.logic.ParityGenerator16`

Defined in bitwise/logic/PAR.py.

16-bit even parity generator.

13.30.2 __init__

```
__init__(
    input_bus,
    parity_bit
)
```

Construct a new 16-bit even parity generator.

Args:

- `input_bus`: An object of type `Bus16`. The input to the parity generator.
- `parity_bit`: An object of type `Wire`. The parity bit.

Raises:

- `TypeError`: If `input_bus` is not a bus of width 16.

13.30.3 __str__

Print out the wire values of the 16-bit even parity generator.

```
input_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
parity_bit: 0
```

13.30.4 __call__

```
__call__(
    input_bus=None,
    parity_bit=None
)
```

Force specific values on the wires of the 16-bit even parity generator.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

CHAPTER 14

Processor

14.1 ArithmeticLogicUnit

14.1.1 Class `bw.processor.ArithmeticLogicUnit`

Defined in bitwise/processor/ALU.py.

16-bit arithmetic logic unit, with functions defined below.

14.1.2 __init__

```
__init__(
    a_bus,
    b_bus,
    function_select_bus,
    overflow,
    carry_out,
    output_bus
)
```

Construct a new 16-bit arithmetic-logic unit with the following functions:

```
0000: a
0001: NOT a
0010: b
0011: NOT b
0100: a AND b
0101: a NAND b
0110: a OR b
0111: a NOR b
1000: a XOR b
```

(continues on next page)

(continued from previous page)

```
1001: a XNOR b
1010: a PLUS b
1011: NOT (a PLUS b)
1100: a MINUS b
1101: NOT (a MINUS b)
1110: 0
1111: 1
```

Args:

- `a_bus`: An object of type `Bus16`. The first input to the ALU. The first addend in add operations and the minuend in subtract operations. Also the number to be compared. `a_bus[0]` and `a_bus[15]` are the most and least significant bit, respectively.
- `b_bus`: An object of type `Bus16`. The second input to the ALU. The second addend in add operations and the subtrahend in subtract operations. Also the number to be compared against. `b_bus[0]` and `b_bus[15]` are the most and least significant bit, respectively.
- `function_select_bus`: An object of type `Bus4`. The function select input of the ALU, with functions defined above. `function_select_bus[0]` and `function_select_bus[3]` are the most and least significant bit, respectively.
- `overflow`: An object of type `Wire`. The arithmetic overflow indicator. Only valid for functions `1100` and `1101` (subtract operations).
- `carry_out`: An object of type `Wire`. The carry-out. Only valid for functions `1010` and `1011` (add operations).
- `output_bus`: An object of type `Bus16`. The output of the ALU. `output_bus[0]` and `output_bus[15]` are the most and least significant bit, respectively.

Raises:

- `TypeError`: If either `a_bus`, `b_bus`, or `output_bus` is not a bus of width 16, or if `fn_select_bus` is not a bus of width 4.

14.1.3 __str__

Print out the wire values of the ALU.

```
a_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
b_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
function_select_bus: (0, 0, 0, 0)
overflow: 0
carry_out: 0
output_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
```

14.1.4 __call__

```
__call__(
    a_bus=None,
    b_bus=None,
    function_select_bus=None,
    overflow=None,
    carry_out=None,
    output_bus=None
)
```

Force specific values on the wires of the ALU.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

14.2 ConditionCodeFlags

14.2.1 Class `bw.processor.ConditionCodeFlags`

Defined in bitwise/processor/FLAG.py.

Condition code flag flip-flops.

14.2.2 __init__

```
__init__(
    data_bus,
    overflow,
    carry_out,
    enable,
    clock,
    z,
    v,
    n,
    c
)
```

Construct a new set of condition code flag flip-flops.

Args:

- `data_bus`: An object of type `Bus16`. The data input to the flip-flops.
- `overflow`: An object of type `Wire`. The overflow input.
- `carry_out`: An object of type `Wire`. The carry-out input.
- `enable`: An object of type `Wire`. The enable input.
- `clock`: An object of type `Wire` or `Clock`. The clock input to the flip-flops.
- `z`: An object of type `Wire`. Indicates when the value on `data_bus` is equal to zero.
- `v`: An object of type `Wire`. Indicates when an arithmetic operation produces an overflow.
- `n`: An object of type `Wire`. Indicates when the value on `data_bus` is negative.

- `c`: An object of type `Wire`. Indicates when an arithmetic operation produces a carry-out.

Raises:

- `TypeError`: If `data_bus` is not a bus of width 16.

14.2.3 __str__

Print out the wire values of the condition code flag flip-flops.

```
data_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
overflow: 0
carry_out: 0
enable: 0
clock: 0
z: 0
v: 0
n: 0
c: 0
```

14.2.4 __call__

```
__call__(
    data_bus=None,
    overflow=None,
    carry_out=None,
    enable=None,
    clock=None,
    z=None,
    v=None,
    n=None,
    c=None
)
```

Force specific values on the wires of the condition code flag flip-flops.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

14.3 ProgramCounter

14.3.1 Class `bw.processor.ProgramCounter`

Defined in bitwise/processor/PC.py.

16-bit program counter.

14.3.2 __init__

```
__init__(
    data_bus,
    up,
    load,
    clock,
    output_bus
)
```

Construct a new program counter with a 16-bit address space.

Args:

- `data_bus`: An object of type `Bus16`.
- `up`: An object of type `Wire`. If its value is 1, increments the program counter on the positive clock edge.
- `load`: An object of type `Wire`. If its value is 1, loads the value of `data_bus` into the program counter on the positive clock edge. If both up and load have value 1, `load` takes precedence.
- `clock`: An object of type `Wire` or `Clock`. The clock input.
- `output_bus`: An object of type `Bus16`. The address of the instruction to be executed.

Raises:

- `TypeError`: If either `data_bus` or `output_bus` is not a bus of width 16.

14.3.3 __str__

Print out the wire values of the program counter.

```
data_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
up: 0
load: 0
clock: 0
output_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
```

14.3.4 __call__

```
__call__(
    data_bus=None,
    up=None,
    load=None,
    clock=None,
    output_bus=None
)
```

Force specific values on the wires of the program counter.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

14.4 StackPointer

14.4.1 Class `bw.processor.StackPointer`

Defined in bitwise/processor/SP.py.

16-bit stack pointer.

14.4.2 __init__

```
__init__(
    up,
    down,
    clock,
    output_bus
)
```

Construct a new stack pointer to a 16-bit address space.

Args:

- `up`: An object of type `Wire`. If its value is 1, increments the stack pointer on the positive clock edge.
- `down`: An object of type `Wire`. If its value is 1, decrements the stack pointer on the positive clock edge. If both `up` and `down` have value 1, `down` takes precedence.
- `clock`: An object of type `Wire` or `Clock`. The clock input.
- `output_bus`: An object of type `Bus16`. The address on top of the stack.

Raises:

- `TypeError`: If `output_bus` is not a bus of width 16.

14.4.3 __str__

Print out the wire values of the stack pointer.

```
up: 0
down: 0
clock: 0
output_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
```

14.4.4 __call__

```
__call__(
    up=None,
    down=None,
    clock=None,
```

(continues on next page)

(continued from previous page)

```
    output_bus=None
)
```

Force specific values on the wires of the stack pointer.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

CHAPTER 15

Signal

15.1 ControlledInverter4

15.1.1 Class `bw.signal.ControlledInverter4`

Defined in bitwise/signal/INV_CTRL.py.

4-bit controlled inverter.

15.1.2 __init__

```
__init__(
    enable,
    input_bus,
    output_bus
)
```

Construct a new 4-bit controlled inverter.

Args:

- `enable`: An object of type `Wire`. Enables the controlled inverter.
- `input_bus`: An object of type `Bus4`. The data input to the controlled inverter.
- `output_bus`: An object of type `Bus4`. The output of the controlled inverter, which is the inverted form of the data input iff `enable` has value 1.

Raises:

- `TypeError`: If either `input_bus` or `output_bus` is not a bus of width 4.

15.1.3 __str__

Print out the wire values of the 4-bit controlled inverter.

```
enable: 0
input_bus: (0, 0, 0, 0)
output_bus: (0, 0, 0, 0)
```

15.1.4 __call__

```
__call__(
    enable=None,
    input_bus=None,
    output_bus=None
)
```

Force specific values on the wires of the 4-bit controlled inverter.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

15.2 ControlledInverter8

15.2.1 Class `bw.signal.ControlledInverter8`

Defined in bitwise/signal/INV_CTRL.py.

8-bit controlled inverter.

15.2.2 __init__

```
__init__(
    enable,
    input_bus,
    output_bus
)
```

Construct a new 8-bit controlled inverter.

Args:

- `enable`: An object of type `Wire`. Enables the controlled inverter.
- `input_bus`: An object of type `Bus8`. The data input to the controlled inverter.
- `output_bus`: An object of type `Bus8`. The output of the controlled inverter, which is the inverted form of the data input iff `enable` has value 1.

Raises:

- `TypeError`: If either `input_bus` or `output_bus` is not a bus of width 8.

15.2.3 __str__

Print out the wire values of the 8-bit controlled inverter.

```
enable: 0
input_bus: (0, 0, 0, 0, 0, 0, 0, 0)
output_bus: (0, 0, 0, 0, 0, 0, 0, 0)
```

15.2.4 __call__

```
__call__(
    enable=None,
    input_bus=None,
    output_bus=None
)
```

Force specific values on the wires of the 8-bit controlled inverter.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

15.3 ControlledInverter16

15.3.1 Class `bw.signal.ControlledInverter16`

Defined in bitwise/signal/INV_CTRL.py.

16-bit controlled inverter.

15.3.2 __init__

```
__init__(
    enable,
    input_bus,
    output_bus
)
```

Construct a new 16-bit controlled inverter.

Args:

- `enable`: An object of type `Wire`. Enables the controlled inverter.
- `input_bus`: An object of type `Bus16`. The data input to the controlled inverter.
- `output_bus`: An object of type `Bus16`. The output of the controlled inverter, which is the inverted form of the data input iff `enable` has value 1.

Raises:

- `TypeError`: If either `input_bus` or `output_bus` is not a bus of width 16.

15.3.3 __str__

Print out the wire values of the 16-bit controlled inverter.

```
enable: 0
input_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
output_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
```

15.3.4 __call__

```
__call__(
    enable=None,
    input_bus=None,
    output_bus=None
)
```

Force specific values on the wires of the 16-bit controlled inverter.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

15.4 Decoder1Of4

15.4.1 Class `bw.signal.Decoder1Of4`

Defined in bitwise/signal/DEC.py.

1-of-4 decoder.

15.4.2 __init__

```
__init__(
    enable,
    input_1,
    input_2,
    output_bus
)
```

Construct a new 1-of-4 decoder.

Args:

- `enable`: An object of type `Wire`. Enables the decoder.
- `input_1`: An object of type `Wire`. The most significant bit of the data input.

- `input_2`: An object of type `Wire`. The least significant bit of the data input.
- `output_bus`: An object of type `Bus4`. A one-hot encoded value of the input, with `output_bus[0]` corresponding to a (1, 1) input and `output_bus[3]` corresponding to a (0, 0) input.

Raises:

- `TypeError`: If `output_bus` is not a bus of width 4.

15.4.3 __str__

Print out the wire values of the 1-of-4 decoder.

```
enable: 0
input_1: 0
input_2: 0
output_bus: (0, 0, 0, 0)
```

15.4.4 __call__

```
__call__(
    enable=None,
    input_1=None,
    input_2=None,
    output_bus=None
)
```

Force specific values on the wires of the 1-of-4 decoder.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

15.5 Decoder1Of8

15.5.1 Class `bw.signal.Decoder1Of8`

Defined in bitwise/signal/DEC.py.

1-of-8 decoder.

15.5.2 __init__

```
__init__(
    enable,
    input_1,
    input_2,
    input_3,
    output_bus
)
```

Construct a new 1-of-8 decoder.

Args:

- `enable`: An object of type `Wire`. Enables the decoder.
- `input_1`: An object of type `Wire`. The most significant bit of the data input.
- `input_2`: An object of type `Wire`.
- `input_3`: An object of type `Wire`. The least significant bit of the data input.
- `output_bus`: An object of type `Bus8`. A one-hot encoded value of the data input, with `output_bus[0]` corresponding to a (1, 1, 1) input and `output_bus[7]` corresponding to a (0, 0, 0) input.

Raises:

- `TypeError`: If `output_bus` is not a bus of width 8.

15.5.3 __str__

Print out the wire values of the 1-of-8 decoder.

```
enable: 0
input_1: 0
input_2: 0
input_3: 0
output_bus: (0, 0, 0, 0, 0, 0, 0, 0)
```

15.5.4 __call__

```
__call__(
    enable=None,
    input_1=None,
    input_2=None,
    input_3=None,
    output_bus=None
)
```

Force specific values on the wires of the 1-of-8 decoder.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

15.6 Decoder1Of16

15.6.1 Class `bw.signal.Decoder1Of16`

Defined in bitwise/signal/DEC.py.

1-of-16 decoder.

15.6.2 __init__

```
__init__(
    enable,
    input_bus,
    output_bus
)
```

Construct a new 1-of-16 decoder.

Args:

- `enable`: An object of type `Wire`. Enables the decoder.
- `input_bus`: An object of type `Bus4`. The data input to the decoder. `input_bus[0]` and `input_bus[3]` are the most and least significant bit, respectively.
- `output_bus`: An object of type `Bus16`. A one-hot encoded value of the input, with `output_bus[0]` corresponding to a (1, 1, 1, 1) input and `output_bus[15]` corresponding to a (0, 0, 0, 0) input.

Raises:

- `TypeError`: If `input_bus` is not a bus of width 4, or if `output_bus` is not a bus of width 16.

15.6.3 __str__

Print out the wire values of the 1-of-16 decoder.

```
enable: 0
input_bus: (0, 0, 0, 0)
output_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
```

15.6.4 __call__

```
__call__(
    enable=None,
    input_bus=None,
    output_bus=None
)
```

Force specific values on the wires of the 1-of-16 decoder.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

15.7 Demultiplexer1To2

15.7.1 Class `bw.signal.Demultiplexer1To2`

Defined in bitwise/signal/DEMUX.py.

1-to-2 demultiplexer.

15.7.2 __init__

```
__init__(
    enable,
    select,
    input,
    output_1,
    output_2
)
```

Construct a new 1-to-2 demultiplexer.

Args:

- `enable`: An object of type `Wire`. Enables the demultiplexer.
- `select`: An object of type `Wire`. The select input.
- `input`: An object of type `Wire`. The data input to the demultiplexer.
- `output_1`: An object of type `Wire`. Takes on the value of `input` if the value of `select` is 1.
- `output_2`: An object of type `Wire`. Takes on the value of `input` if the value of `select` is 0.

15.7.3 __str__

Print out the wire values of the 1-to-2 demultiplexer.

```
enable: 0
select: 0
input: 0
output_1: 0
output_2: 0
```

15.7.4 __call__

```
__call__(
    enable=None,
    select=None,
    input=None,
    output_1=None,
    output_2=None
)
```

Force specific values on the wires of the 1-to-2 demultiplexer.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

15.8 Demultiplexer1To4

15.8.1 Class `bw.signal.Demultiplexer1To4`

Defined in bitwise/signal/DEMUX.py.

1-to-4 demultiplexer.

15.8.2 __init__

```
__init__(
    enable,
    select_1,
    select_2,
    input,
    output_bus
)
```

Construct a new 1-to-4 demultiplexer.

Args:

- `enable`: An object of type `Wire`. Enables the demultiplexer.
- `select_1`: An object of type `Wire`. The most significant bit of the select input.
- `select_2`: An object of type `Wire`. The least significant bit of the select input.
- `input`: An object of type `Wire`. The data input to the demultiplexer.
- `output_bus`: An object of type `Bus4`. `output_bus[0]` takes on the value of `input` for a (1, 1) select, and `output_bus[3]` takes on the value of `input` for a (0, 0) select.

Raises:

- `TypeError`: If `output_bus` is not a bus of width 4.

15.8.3 __str__

Print out the wire values of the 1-to-4 demultiplexer.

```
enable: 0
select_1: 0
select_2: 0
input: 0
output_bus: (0, 0, 0, 0)
```

15.8.4 __call__

```
__call__(
    enable=None,
    select_1=None,
    select_2=None,
    input=None,
    output_bus=None
)
```

Force specific values on the wires of the 1-to-4 demultiplexer.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

15.9 Demultiplexer1To8

15.9.1 Class `bw.signal.Demultiplexer1To8`

Defined in bitwise/signal/DEMUX.py.

1-to-8 demultiplexer.

15.9.2 __init__

```
__init__(
    enable,
    select_1,
    select_2,
    select_3,
    input,
    output_bus
)
```

Construct a new 1-to-8 demultiplexer.

Args:

- `enable`: An object of type `Wire`. Enables the demultiplexer.
- `select_1`: An object of type `Wire`. The most significant bit of the select input.
- `select_2`: An object of type `Wire`.
- `select_3`: An object of type `Wire`. The least significant bit of the select input.
- `input`: An object of type `Wire`. The data input to the demultiplexer.
- `output_bus`: An object of type `Bus8`. `output_bus[0]` takes on the value of `input` for a (1, 1, 1) select, and `output_bus[7]` takes on the value of `input` for a (0, 0, 0) select.

Raises:

- `TypeError`: If `output_bus` is not a bus of width 8.

15.9.3 __str__

Print out the wire values of the 1-to-8 demultiplexer.

```
enable: 0
select_1: 0
select_2: 0
select_3: 0
input: 0
output_bus: (0, 0, 0, 0, 0, 0, 0, 0)
```

15.9.4 __call__

```
__call__(
    enable=None,
    select_1=None,
    select_2=None,
    select_3=None,
    input=None,
    output_bus=None
)
```

Force specific values on the wires of the 1-to-8 demultiplexer.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

15.10 Demultiplexer1To16

15.10.1 Class `bw.signal.Demultiplexer1To16`

Defined in bitwise/signal/DEMUX.py.

1-to-16 demultiplexer.

15.10.2 __init__

```
__init__(
    enable,
    select_bus,
    input,
    output_bus
)
```

Construct a new 1-to-16 demultiplexer.

Args:

- `enable`: An object of type `Wire`. Enables the demultiplexer.

- `select_bus`: An object of type `Bus4`. The select input to the demultiplexer. `select_bus[0]` and `select_bus[3]` are the most and least significant bit, respectively.
- `input`: An object of type `Wire`. The data input to the demultiplexer.
- `output_bus`: An object of type `Bus16`. `output_bus[0]` takes on the value of `input` for a (1, 1, 1, 1) select, and `output_bus[15]` takes on the value of `input` for a (0, 0, 0, 0) select.

Raises:

- `TypeError`: If `select_bus` is not a bus of width 4, or if `output_bus` is not a bus of width 16.

15.10.3 __str__

Print out the wire values of the 1-to-16 demultiplexer.

```
enable: 0
select_bus: (0, 0, 0, 0)
input: 0
output_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
```

15.10.4 __call__

```
__call__(
    enable=None,
    select_bus=None,
    input=None,
    output_bus=None
)
```

Force specific values on the wires of the 1-to-16 demultiplexer.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

15.11 Encoder4To2

15.11.1 Class `bw.signal.Encoder4To2`

Defined in bitwise/signal/ENC.py.

4-to-2 priority encoder.

15.11.2 __init__

```
__init__(
    enable,
    input_bus,
    valid,
    output_1,
```

(continues on next page)

(continued from previous page)

```
    output_2
)
```

Construct a new 4-to-2 priority encoder.

Args:

- `enable`: An object of type `Wire`. Enables the encoder.
- `input_bus`: An object of type `Bus4`. The data input to the encoder. `input_bus[0]` corresponds to an input value of 3, and `input_bus[3]` corresponds to an input value of 0.
- `valid`: An object of type `Wire`. The valid indicator. Only takes on the value 0 if all the wires in `input_bus` have value 0.
- `output_1`: An object of type `Wire`. The most significant bit of the output.
- `output_2`: An object of type `Wire`. The least significant bit of the output.

Raises:

- `TypeError`: If `input_bus` is not a bus of width 4.

15.11.3 __str__

Print out the wire values of the 4-to-2 priority encoder.

```
enable: 0
input_bus: (0, 0, 0, 0)
valid: 0
output_1: 0
output_2: 0
```

15.11.4 __call__

```
__call__(
    enable=None,
    input_bus=None,
    valid=None,
    output_1=None,
    output_2=None
)
```

Force specific values on the wires of the 4-to-2 priority encoder.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

15.12 Encoder8To3

15.12.1 Class `bw.signal.Encoder8To3`

Defined in bitwise/signal/ENC.py.

8-to-3 priority encoder.

15.12.2 __init__

```
__init__(
    enable,
    input_bus,
    valid,
    output_1,
    output_2,
    output_3
)
```

Construct a new 8-to-3 priority encoder.

Args:

- `enable`: An object of type `Wire`. Enables the encoder.
- `input_bus`: An object of type `Bus8`. The data input to the encoder. `input_bus[0]` corresponds to an input value of 7, and `input_bus[7]` corresponds to an input value of 0.
- `valid`: An object of type `Wire`. The valid indicator. Only takes on the value 0 if all the wires in `input_bus` have value 0.
- `output_1`: An object of type `Wire`. The most significant bit of the output.
- `output_2`: An object of type `Wire`.
- `output_3`: An object of type `Wire`. The least significant bit of the output.

Raises:

- `TypeError`: If `input_bus` is not a bus of width 8.

15.12.3 __str__

Print out the wire values of the 8-to-3 priority encoder.

```
enable: 0
input_bus: (0, 0, 0, 0, 0, 0, 0, 0)
valid: 0
output_1: 0
output_2: 0
output_3: 0
```

15.12.4 __call__

```
__call__(
    enable=None,
    input_bus=None,
    valid=None,
    output_1=None,
    output_2=None,
    output_3=None
)
```

Force specific values on the wires of the 8-to-3 priority encoder.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

15.13 Encoder16To4

15.13.1 Class `bw.signal.Encoder16To4`

Defined in bitwise/signal/ENC.py.

16-to-4 priority encoder.

15.13.2 __init__

```
__init__(
    enable,
    input_bus,
    valid,
    output_bus
)
```

Construct a new 16-to-4 priority encoder.

Args:

- `enable`: An object of type `Wire`. Enables the encoder.
- `input_bus`: An object of type `Bus16`. The data input to the encoder. `input_bus[0]` corresponds to an input value of 15, and `input_bus[15]` corresponds to an input value of 0.
- `valid`: An object of type `Wire`. The valid indicator. Only takes on the value 0 if all the wires in `input_bus` have value 0.
- `output_bus`: An object of type `Bus4`. The output of the encoder. `output_bus[0]` and `output_bus[3]` are the most and least significant bit, respectively.

Raises:

- `TypeError`: If `input_bus` is not a bus of width 16, or if `output_bus` is not a bus of width 4.

15.13.3 __str__

Print out the wire values of the 16-to-4 priority encoder.

```
enable: 0
input_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
valid: 0
output_bus: (0, 0, 0, 0)
```

15.13.4 __call__

```
__call__(
    enable=None,
    input_bus=None,
    valid=None,
    output_bus=None
)
```

Force specific values on the wires of the 16-to-4 priority encoder.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

15.14 Multiplexer2To1

15.14.1 Class `bw.signal.Multiplexer2To1`

Defined in bitwise/signal/MUX.py.

2-to-1 multiplexer.

15.14.2 __init__

```
__init__(
    enable,
    select,
    input_1,
    input_2,
    output
)
```

Construct a new 2-to-1 multiplexer.

Args:

- `enable`: An object of type `Wire`. Enables the multiplexer.
- `select`: An object of type `Wire`. The select input.
- `input_1`: An object of type `Wire`. The first data input to the multiplexer.
- `input_2`: An object of type `Wire`. The second data input to the multiplexer.

- `output`: An object of type `Wire`. The output of the multiplexer. Takes on the value of `input_1` for a 1 select and `input_2` for a 0 select.

15.14.3 __str__

Print out the wire values of the 2-to-1 multiplexer.

```
enable: 0
select: 0
input_1: 0
input_2: 0
output: 0
```

15.14.4 __call__

```
__call__(
    enable=None,
    select=None,
    input_1=None,
    input_2=None,
    output=None
)
```

Force specific values on the wires of the 2-to-1 multiplexer.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

15.15 Multiplexer4To1

15.15.1 Class `bw.signal.Multiplexer4To1`

Defined in bitwise/signal/MUX.py.

4-to-1 multiplexer.

15.15.2 __init__

```
__init__(
    enable,
    select_1,
    select_2,
    input_bus,
    output
)
```

Construct a new 4-to-1 multiplexer.

Args:

- `enable`: An object of type `Wire`. Enables the multiplexer.
- `select_1`: An object of type `Wire`. The most significant bit of the select input.
- `select_2`: An object of type `Wire`. The least significant bit of the select input.
- `input_bus`: An object of type `Bus4`. The data input to the multiplexer.
- `output`: An object of type `Wire`. The output of the multiplexer. Takes on the value of `input_bus[0]` for a (1, 1) select and `input_bus[3]` for a (0, 0) select.

Raises:

- `TypeError`: If `input_bus` is not a bus of width 4.

15.15.3 __str__

Print out the wire values of the 4-to-1 multiplexer.

```
enable: 0
select_1: 0
select_2: 0
input_bus: (0, 0, 0, 0)
output: 0
```

15.15.4 __call__

```
__call__(
    enable=None,
    select_1=None,
    select_2=None,
    input_bus=None,
    output=None
)
```

Force specific values on the wires of the 4-to-1 multiplexer.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

15.16 Multiplexer8To1

15.16.1 Class `bw.signal.Multiplexer8To1`

Defined in bitwise/signal/MUX.py.

8-to-1 multiplexer.

15.16.2 __init__

```
__init__(
    enable,
    select_1,
    select_2,
    select_3,
    input_bus,
    output
)
```

Construct a new 8-to-1 multiplexer.

Args:

- `enable`: An object of type `Wire`. Enables the multiplexer.
- `select_1`: An object of type `Wire`. The most significant bit of the select input.
- `select_2`: An object of type `Wire`.
- `select_3`: An object of type `Wire`. The least significant bit of the select input.
- `input_bus`: An object of type `Bus8`. The data input to the multiplexer.
- `output`: An object of type `Wire`. The output of the multiplexer. Takes on the value of `input_bus[0]` for a (1, 1, 1) select and `input_bus[7]` for a (0, 0, 0) select.

Raises:

- `TypeError`: If `input_bus` is not a bus of width 8.

15.16.3 __str__

Print out the wire values of the 8-to-1 multiplexer.

```
enable: 0
select_1: 0
select_2: 0
select_3: 0
input_bus: (0, 0, 0, 0, 0, 0, 0, 0)
output: 0
```

15.16.4 __call__

```
__call__(
    enable=None,
    select_1=None,
    select_2=None,
    select_3=None,
    input_bus=None,
    output=None
)
```

Force specific values on the wires of the 8-to-1 multiplexer.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

15.17 Multiplexer16To1

15.17.1 Class `bw.signal.Multiplexer16To1`

Defined in bitwise/signal/MUX.py.

16-to-1 multiplexer.

15.17.2 __init__

```
__init__(
    enable,
    select_bus,
    input_bus,
    output
)
```

Construct a new 16-to-1 multiplexer.

Args:

- `enable`: An object of type `Wire`. Enables the multiplexer.
- `select_bus`: An object of type `Bus4`. `select_bus[0]` and `select_bus[3]` are the most and least significant bit, respectively.
- `input_bus`: An object of type `Bus16`. The data input to the multiplexer.
- `output`: An object of type `Wire`. The output of the multiplexer. Takes on the value of `input_bus[0]` for a (1, 1, 1, 1) select and `input_bus[15]` for a (0, 0, 0, 0) select.

Raises:

- `TypeError`: If `select_bus` is not a bus of width 4, or if `input_bus` is not a bus of width 16.

15.17.3 __str__

Print out the wire values of the 16-to-1 multiplexer.

```
enable: 0
select_bus: (0, 0, 0, 0)
input_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
output: 0
```

15.17.4 __call__

```
__call__(
    enable=None,
    select_bus=None,
    input_bus=None,
    output=None
)
```

Force specific values on the wires of the 16-to-1 multiplexer.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

15.18 SevenSegmentConverter

15.18.1 Class `bw.signal.SevenSegmentConverter`

Defined in bitwise/signal/SSD.py.

Seven-segment converter with a common anode.

15.18.2 __init__

```
__init__(
    enable,
    input_bus,
    output_bus
)
```

Construct a new seven-segment converter.

Args:

- `enable`: An object of type `Wire`. Enables the seven-segment converter.
- `input_bus`: An object of type `Bus4`. The data input to the seven-segment converter. `input_bus[0]` and `input_bus[3]` are the most and least significant bit, respectively.
- `output_bus`: An object of type `BusSevenSegmentDisplay`. The output of the seven-segment converter. `output_bus[0]` and `output_bus[7]` correspond to segment G and segment A, respectively.

Raises:

- `TypeError`: If `input_bus` is not a bus of width 4, or if `output_bus` is not a bus of width 7.

15.18.3 __str__

Print out the wire values of the seven-segment converter.

```
enable: 0
input_bus: (0, 0, 0, 0)
output_bus: (0, 0, 0, 0, 0, 0, 0)
```

15.18.4 __call__

```
__call__(
    enable=None,
    input_bus=None,
    output_bus=None
)
```

Force specific values on the wires of the seven-segment converter.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

15.19 SevenSegmentConverterDual

15.19.1 Class `bw.signal.SevenSegmentConverterDual`

Defined in bitwise/signal/SSD.py.

Dual seven-segment converter with a common anode.

15.19.2 __init__

```
__init__(
    enable,
    input_bus,
    output_bus_1,
    output_bus_2
)
```

Construct a new dual seven-segment converter.

Args:

- `enable`: An object of type `Wire`. Enables the seven-segment converter.
- `input_bus`: An object of type `Bus8`. The data input to the seven-segment converter. `input_bus[0]` and `input_bus[7]` are the most and least significant bit, respectively.
- `output_bus_1`: An object of type `BusSevenSegmentDisplay`. The first output bus of the seven-segment converter, using `input_bus[0]` and `input_bus[3]` as the most and least significant bit, respectively. `output_bus_1[0]` and `output_bus_1[7]` correspond to segment G and segment A, respectively.
- `output_bus_2`: An object of type `BusSevenSegmentDisplay`. The second output bus of the seven-segment converter, using `input_bus[4]` and `input_bus[7]` as the most and least significant bit, respectively. `output_bus_2[0]` and `output_bus_2[7]` correspond to segment G and segment A, respectively.

Raises:

- `TypeError`: If `input_bus` is not a bus of width 8, or if either `output_bus_1` or `output_bus_2` is not a bus of width 7.

15.19.3 __str__

Print out the wire values of the dual seven-segment converter.

```
enable: 0
input_bus: (0, 0, 0, 0, 0, 0, 0, 0)
output_bus_1: (0, 0, 0, 0, 0, 0, 0)
output_bus_2: (0, 0, 0, 0, 0, 0, 0)
```

15.19.4 __call__

```
__call__(
    enable=None,
    input_bus=None,
    output_bus_1=None,
    output_bus_2=None
)
```

Force specific values on the wires of the dual seven-segment converter.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

15.20 SevenSegmentConverterQuad

15.20.1 Class `bw.signal.SevenSegmentConverterQuad`

Defined in bitwise/signal/SSD.py.

Quad seven-segment converter with a common anode.

15.20.2 __init__

```
__init__(
    enable,
    input_bus,
    output_bus_1,
    output_bus_2,
    output_bus_3,
    output_bus_4
)
```

Construct a new quad seven-segment converter.

Args:

- `enable`: An object of type `Wire`. Enables the seven-segment converter.
- `input_bus`: An object of type `Bus16`. The data input to the seven-segment converter. `input_bus[0]` and `input_bus[15]` are the most and least significant bit, respectively.
- `output_bus_1`: An object of type `BusSevenSegmentDisplay`. The first output bus of the seven-segment converter, using `input_bus[0]` and `input_bus[3]` as the most and least significant bit, respectively. `output_bus_1[0]` and `output_bus_1[7]` correspond to segment G and segment A, respectively.
- `output_bus_2`: An object of type `BusSevenSegmentDisplay`. The second output bus of the seven-segment converter, using `input_bus[4]` and `input_bus[7]` as the most and least significant bit, respectively. `output_bus_2[0]` and `output_bus_2[7]` correspond to segment G and segment A, respectively.
- `output_bus_3`: An object of type `BusSevenSegmentDisplay`. The third output bus of the seven-segment converter, using `input_bus[8]` and `input_bus[11]` as the most and least significant bit, respectively. `output_bus_3[0]` and `output_bus_3[7]` correspond to segment G and segment A, respectively.
- `output_bus_4`: An object of type `BusSevenSegmentDisplay`. The fourth output bus of the seven-segment converter, using `input_bus[12]` and `input_bus[15]` as the most and least significant bit, respectively. `output_bus_4[0]` and `output_bus_4[7]` correspond to segment G and segment A, respectively.

Raises:

- `TypeError`: If `input_bus` is not a bus of width 16, or if either `output_bus_1`, `output_bus_2`, `output_bus_3`, or `output_bus_4` is not a bus of width 7.

15.20.3 __str__

Print out the wire values of the quad seven-segment converter.

```
enable: 0
input_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
output_bus_1: (0, 0, 0, 0, 0, 0, 0)
output_bus_2: (0, 0, 0, 0, 0, 0, 0)
output_bus_3: (0, 0, 0, 0, 0, 0, 0)
output_bus_4: (0, 0, 0, 0, 0, 0, 0)
```

15.20.4 __call__

```
__call__(
    enable=None,
    input_bus=None,
    output_bus_1=None,
    output_bus_2=None,
    output_bus_3=None,
    output_bus_4=None
)
```

Force specific values on the wires of the quad seven-segment converter.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

CHAPTER 16

State

16.1 DownCounterMod4

16.1.1 Class `bw.state.DownCounterMod4`

Defined in bitwise/state/COUNT.py.

2-bit (mod-4) down counter with parallel load.

16.1.2 __init__

```
__init__(
    enable,
    load_n,
    load_1,
    load_2,
    clock,
    output_1,
    output_2
)
```

Construct a new mod-4 down counter.

Args:

- `enable`: An object of type `Wire`. Enables the counter.
- `load_n`: An object of type `Wire`. Loads `load_1` into `output_1` and `load_2` into `output_2` if its value is 0.
- `load_1`: An object of type `Wire`. The most significant bit of the load input.

- `load_2`: An object of type `Wire`. The least significant bit of the load input.
- `clock`: An object of type `Wire` or `Clock`. The clock input to the counter.
- `output_1`: An object of type `Wire`. The most significant bit of the output.
- `output_2`: An object of type `Wire`. The least significant bit of the output.

16.1.3 __str__

Print out the wire values of the mod-4 down counter.

```
enable: 0
load_n: 0
load_1: 0
load_2: 0
clock: 0
output_1: 0
output_2: 0
```

16.1.4 __call__

```
__call__(
    enable=None,
    load_n=None,
    load_1=None,
    load_2=None,
    clock=None,
    output_1=None,
    output_2=None
)
```

Force specific values on the wires of the mod-4 down counter.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

16.2 DownCounterMod8

16.2.1 Class `bw.state.DownCounterMod8`

Defined in bitwise/state/COUNT.py.

3-bit (mod-8) down counter with parallel load.

16.2.2 __init__

```
__init__(
    enable,
    load_n,
    load_1,
```

(continues on next page)

(continued from previous page)

```
    load_2,
    load_3,
    clock,
    output_1,
    output_2,
    output_3
)
```

Construct a new mod-8 down counter.

Args:

- `enable`: An object of type `Wire`. Enables the counter.
- `load_n`: An object of type `Wire`. Loads `load_1` into `output_1`, `load_2` into `output_2`, and `load_3` into `output_3` if its value is 0.
- `load_1`: An object of type `Wire`. The most significant bit of the load input.
- `load_2`: An object of type `Wire`.
- `load_3`: An object of type `Wire`. The least significant bit of the load input.
- `clock`: An object of type `Wire` or `Clock`. The clock input to the counter.
- `output_1`: An object of type `Wire`. The most significant bit of the output.
- `output_2`: An object of type `Wire`.
- `output_3`: An object of type `Wire`. The least significant bit of the output.

16.2.3 __str__

Print out the wire values of the mod-8 down counter.

```
enable: 0
load_n: 0
load_1: 0
load_2: 0
load_3: 0
clock: 0
output_1: 0
output_2: 0
output_3: 0
```

16.2.4 __call__

```
__call__(
    enable=None,
    load_n=None,
    load_1=None,
    load_2=None,
    load_3=None,
    clock=None,
    output_1=None,
```

(continues on next page)

(continued from previous page)

```
    output_2=None,
    output_3=None
)
```

Force specific values on the wires of the mod-8 down counter.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

16.3 DownCounterMod16

16.3.1 Class `bw.state.DownCounterMod16`

Defined in bitwise/state/COUNT.py.

4-bit (mod-16) down counter with parallel load.

16.3.2 __init__

```
__init__(
    enable,
    load_n,
    load_bus,
    clock,
    output_bus
)
```

Construct a new mod-16 down counter.

Args:

- `enable`: An object of type `Wire`. Enables the counter.
- `load_n`: An object of type `Wire`. Loads `load_bus` into `output_bus` if its value is 0.
- `load_bus`: An object of type `Bus4`. The load input to the counter. `load_bus[0]` and `load_bus[3]` are the most and least significant bit, respectively.
- `clock`: An object of type `Wire` or `Clock`. The clock input to the counter.
- `output_bus`: An object of type `Bus4`. The output of the counter. `output_bus[0]` and `output_bus[3]` are the most and least significant bit, respectively.

Raises:

- `TypeError`: If either `load_bus` or `output_bus` is not a bus of width 4.

16.3.3 __str__

Print out the wire values of the mod-16 down counter.

```
enable: 0
load_n: 0
load_bus: (0, 0, 0, 0)
clock: 0
output_bus: (0, 0, 0, 0)
```

16.3.4 __call__

```
__call__(
    enable=None,
    load_n=None,
    load_bus=None,
    clock=None,
    output_bus=None
)
```

Force specific values on the wires of the mod-16 down counter.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

16.4 ParallelToSerialConverter4To1

16.4.1 Class `bw.state.ParallelToSerialConverter4To1`

Defined in bitwise/state/PISO.py.

4-bit-parallel-to-serial converter.

16.4.2 __init__

```
__init__(
    enable,
    clear_n,
    load_n,
    data_bus,
    clock,
    output
)
```

Construct a new 4-bit-parallel-to-serial converter.

Args:

- `enable`: An object of type `Wire`. Enables the converter.
- `clear_n`: An object of type `Wire`. Clears all 4 internal registers to 0 asynchronously if its value is 0.

- `load_n`: An object of type `Wire`. The mode select. A value of 0 indicates a parallel load operation, where the values of `data_bus` are loaded into the internal registers. A value of 1 indicates a shift-right operation.
- `data_bus`: An object of type `Bus4`. The parallel data input.
- `clock`: An object of type `Wire` or `Clock`. The clock input.
- `output`: An object of type `Wire`. The serial output of the converter. `data_bus[3]` is outputted first, and `data_bus[0]` is outputted last.

Raises:

- `TypeError`: If `data_bus` is not a bus of width 4.

16.4.3 __str__

Print out the wire values of the 4-bit-parallel-to-serial converter.

```
enable: 0
clear_n: 0
load_n: 0
data_bus: (0, 0, 0, 0)
clock: 0
output: 0
```

16.4.4 __call__

```
__call__(
    enable=None,
    clear_n=None,
    load_n=None,
    data_bus=None,
    clock=None,
    output=None
)
```

Force specific values on the wires of the 4-bit-parallel-to-serial converter.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

16.5 ParallelToSerialConverter8To1

16.5.1 Class `bw.state.ParallelToSerialConverter8To1`

Defined in bitwise/state/PISO.py.

8-bit-parallel-to-serial converter.

16.5.2 __init__

```
__init__(
    enable,
    clear_n,
    load_n,
    data_bus,
    clock,
    output
)
```

Construct a new 8-bit-parallel-to-serial converter.

Args:

- `enable`: An object of type `Wire`. Enables the converter.
- `clear_n`: An object of type `Wire`. Clears all 8 internal registers to 0 asynchronously if its value is 0.
- `load_n`: An object of type `Wire`. The mode select. A value of 0 indicates a parallel load operation, where the values of `data_bus` are loaded into the internal registers. A value of 1 indicates a shift-right operation.
- `data_bus`: An object of type `Bus8`. The parallel data input.
- `clock`: An object of type `Wire` or `Clock`. The clock input.
- `output`: An object of type `Wire`. The serial output of the converter. `data_bus[7]` is outputted first, and `data_bus[0]` is outputted last.

Raises:

- `TypeError`: If `data_bus` is not a bus of width 8.

16.5.3 __str__

Print out the wire values of the 8-bit-parallel-to-serial converter.

```
enable: 0
clear_n: 0
load_n: 0
data_bus: (0, 0, 0, 0, 0, 0, 0, 0)
clock: 0
output: 0
```

16.5.4 __call__

```
__call__(
    enable=None,
    clear_n=None,
    load_n=None,
    data_bus=None,
    clock=None,
    output=None
)
```

Force specific values on the wires of the 8-bit-parallel-to-serial converter.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

16.6 ParallelToSerialConverter16To1

16.6.1 Class `bw.state.ParallelToSerialConverter16To1`

Defined in bitwise/state/PISO.py.

16-bit-parallel-to-serial converter.

16.6.2 __init__

```
__init__(
    enable,
    clear_n,
    load_n,
    data_bus,
    clock,
    output
)
```

Construct a new 16-bit-parallel-to-serial converter.

Args:

- `enable`: An object of type `Wire`. Enables the converter.
- `clear_n`: An object of type `Wire`. Clears all 16 internal registers to 0 asynchronously if its value is 0.
- `load_n`: An object of type `Wire`. The mode select. A value of 0 indicates a parallel load operation, where the values of `data_bus` are loaded into the internal registers. A value of 1 indicates a shift-right operation.
- `data_bus`: An object of type `Bus16`. The parallel data input.
- `clock`: An object of type `Wire` or `Clock`. The clock input.
- `output`: An object of type `Wire`. The serial output of the converter. `data_bus[15]` is outputted first, and `data_bus[0]` is outputted last.

Raises:

- `TypeError`: If `data_bus` is not a bus of width 16.

16.6.3 __str__

Print out the wire values of the 16-bit-parallel-to-serial converter.

```
enable: 0
clear_n: 0
load_n: 0
data_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
clock: 0
output: 0
```

16.6.4 __call__

```
__call__(
    enable=None,
    clear_n=None,
    load_n=None,
    data_bus=None,
    clock=None,
    output=None
)
```

Force specific values on the wires of the 16-bit-parallel-to-serial converter.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

16.7 RingCounter4

16.7.1 Class `bw.state.RingCounter4`

Defined in bitwise/state/RING.py.

4-bit straight ring counter.

16.7.2 __init__

```
__init__(
    enable,
    clear_n,
    clock,
    output_bus
)
```

Construct a new 4-bit ring counter.

Args:

- `enable`: An object of type `Wire`. Enables the ring counter.
- `clear_n`: An object of type `Wire`. Clears `output_bus` to (0, 0, 0, 1) (the 0 state) asynchronously if its value is 0.
- `clock`: An object of type `Wire` or `Clock`. The clock input.

- `output_bus`: An object of type `Bus4`. The one-hot output of the ring counter. Starts at (0, 0, 0, 1) and counts up to (1, 0, 0, 0).

Raises:

- `TypeError`: If `output_bus` is not a bus of width 4.

16.7.3 __str__

Print out the wire values of the 4-bit ring counter.

```
enable: 0
clear_n: 0
clock: 0
output_bus: (0, 0, 0, 0)
```

16.7.4 __call__

```
__call__(
    enable=None,
    clear_n=None,
    clock=None,
    output_bus=None
)
```

Force specific values on the wires of the 4-bit ring counter.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

16.8 RingCounter8

16.8.1 Class `bw.state.RingCounter8`

Defined in bitwise/state/RING.py.

8-bit straight ring counter.

16.8.2 __init__

```
__init__(
    enable,
    clear_n,
    clock,
    output_bus
)
```

Construct a new 8-bit ring counter.

Args:

- `enable`: An object of type `Wire`. Enables the ring counter.
- `clear_n`: An object of type `Wire`. Clears `output_bus` to (0, 0, 0, 0, 0, 0, 0, 1) (the 0 state) asynchronously if its value is 0.
- `clock`: An object of type `Wire` or `Clock`. The clock input.
- `output_bus`: An object of type `Bus8`. The one-hot output of the ring counter. Starts at (0, 0, 0, 0, 0, 0, 0, 1) and counts up to (1, 0, 0, 0, 0, 0, 0, 0).

Raises:

- `TypeError`: If `output_bus` is not a bus of width 8.

16.8.3 __str__

Print out the wire values of the 8-bit ring counter.

```
enable: 0
clear_n: 0
clock: 0
output_bus: (0, 0, 0, 0, 0, 0, 0, 0)
```

16.8.4 __call__

```
__call__(
    enable=None,
    clear_n=None,
    clock=None,
    output_bus=None
)
```

Force specific values on the wires of the 8-bit ring counter.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

16.9 RingCounter16

16.9.1 Class `bw.state.RingCounter16`

Defined in bitwise/state/RING.py.

16-bit straight ring counter.

16.9.2 __init__

```
__init__(
    enable,
    clear_n,
    clock,
    output_bus
)
```

Construct a new 16-bit ring counter.

Args:

- `enable`: An object of type `Wire`. Enables the ring counter.
- `clear_n`: An object of type `Wire`. Clears `output_bus` to (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1) (the 0 state) asynchronously if its value is 0.
- `clock`: An object of type `Wire` or `Clock`. The clock input.
- `output_bus`: An object of type `Bus16`. The one-hot output of the ring counter. Starts at (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1) and counts up to (1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0).

Raises:

- `TypeError`: If `output_bus` is not a bus of width 16.

16.9.3 __str__

Print out the wire values of the 16-bit ring counter.

```
enable: 0
clear_n: 0
clock: 0
output_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
```

16.9.4 __call__

```
__call__(
    enable=None,
    clear_n=None,
    clock=None,
    output_bus=None
)
```

Force specific values on the wires of the 16-bit ring counter.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

16.10 SerialToParallelConverter1To4

16.10.1 Class `bw.state.SerialToParallelConverter1To4`

Defined in bitwise/state/SIPO.py.

Serial-to-4-bit-parallel converter.

16.10.2 __init__

```
__init__(
    enable,
    clear_n,
    data,
    clock,
    output_bus
)
```

Construct a new serial-to-4-bit-parallel converter.

Args:

- `enable`: An object of type `Wire`. Enables the converter.
- `clear_n`: An object of type `Wire`. Clears all 4 internal registers to 0 asynchronously if its value is 0.
- `data`: An object of type `Wire`. The serial data input.
- `clock`: An object of type `Wire` or `Clock`. The clock input.
- `output_bus`: An object of type `Bus4`. The parallel output of the converter. `output[0]` corresponds to the most recent serial data input.

Raises:

- `TypeError`: If `output_bus` is not a bus of width 4.

16.10.3 __str__

Print out the wire values of the serial-to-4-bit-parallel converter.

```
enable: 0
clear_n: 0
data: 0
clock: 0
output_bus: (0, 0, 0, 0)
```

16.10.4 __call__

```
__call__(
    enable=None,
    clear_n=None,
    data=None,
    clock=None,
    output_bus=None
)
```

Force specific values on the wires of the serial-to-4-bit-parallel converter.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

16.11 SerialToParallelConverter1To8

16.11.1 Class `bw.state.SerialToParallelConverter1To8`

Defined in bitwise/state/SIPO.py.

Serial-to-8-bit-parallel converter.

16.11.2 __init__

```
__init__(
    enable,
    clear_n,
    data,
    clock,
    output_bus
)
```

Construct a new serial-to-8-bit-parallel converter.

Args:

- `enable`: An object of type `Wire`. Enables the converter.
- `clear_n`: An object of type `Wire`. Clears all 8 internal registers to 0 asynchronously if its value is 0.
- `data`: An object of type `Wire`. The serial data input.
- `clock`: An object of type `Wire` or `Clock`. The clock input.
- `output_bus`: An object of type `Bus8`. The parallel output of the converter. `output[0]` corresponds to the most recent serial data input.

Raises:

- `TypeError`: If `output_bus` is not a bus of width 8.

16.11.3 __str__

Print out the wire values of the serial-to-8-bit-parallel converter.

```
enable: 0
clear_n: 0
data: 0
clock: 0
output_bus: (0, 0, 0, 0, 0, 0, 0, 0)
```

16.11.4 __call__

```
__call__(
    enable=None,
    clear_n=None,
    data=None,
    clock=None,
    output_bus=None
)
```

Force specific values on the wires of the serial-to-8-bit-parallel converter.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

16.12 SerialToParallelConverter1To16

16.12.1 Class `bw.state.SerialToParallelConverter1To16`

Defined in bitwise/state/SIPO.py.

Serial-to-16-bit-parallel converter.

16.12.2 __init__

```
__init__(
    enable,
    clear_n,
    data,
    clock,
    output_bus
)
```

Construct a new serial-to-16-bit-parallel converter.

Args:

- `enable`: An object of type `Wire`. Enables the converter.
- `clear_n`: An object of type `Wire`. Clears all 16 internal registers to 0 asynchronously if its value is 0.
- `data`: An object of type `Wire`. The serial data input.

- `clock`: An object of type `Wire` or `Clock`. The clock input.
- `output_bus`: An object of type `Bus16`. The parallel output of the converter. `output[0]` corresponds to the most recent serial data input.

Raises:

- `TypeError`: If `output_bus` is not a bus of width 16.

16.12.3 __str__

Print out the wire values of the serial-to-16-bit-parallel converter.

```
enable: 0
clear_n: 0
data: 0
clock: 0
output_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
```

16.12.4 __call__

```
__call__(
    enable=None,
    clear_n=None,
    data=None,
    clock=None,
    output_bus=None
)
```

Force specific values on the wires of the serial-to-16-bit-parallel converter.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

16.13 ShiftRegister4

16.13.1 Class `bw.state.ShiftRegister4`

Defined in bitwise/state/SHIFT.py.

4-bit shift register.

16.13.2 __init__

```
__init__(
    enable,
    clear_n,
    shift_load,
    data_bus,
```

(continues on next page)

(continued from previous page)

```
    data_serial,
    clock,
    output_bus,
    output_serial
)
```

Construct a new 4-bit shift register.

Args:

- `enable`: An object of type `Wire`. Enables the shift register.
- `clear_n`: An object of type `Wire`. Clears `output_bus` and `output_serial` to 0 asynchronously if its value is 0.
- `shift_load`: An object of type `Wire`. The mode select. A value of 0 indicates a parallel load operation, where `output_bus` takes on the value of `data_bus`. A value of 1 indicates a shift-right operation, where `output_bus[3]` takes on the value of `output_bus[2]`, `output_bus[2]` takes on the value of `output_bus[1]`, and so on; `output_bus[0]` takes on the value of `data_serial`.
- `data_bus`: An object of type `Bus4`. The parallel data input.
- `data_serial`. An object of type `Wire`. The serial data input.
- `clock`. An object of type `Wire` or `Clock`. The clock input to the shift register.
- `output_bus`. An object of type `Bus4`. The parallel data output.
- `output_serial`. An object of type `Wire`. The serial data output. Identical to `output_bus[3]`.

Raises:

- `TypeError`: If either `data_bus` or `output_bus` is not a bus of width 4.

16.13.3 __str__

Print out the wire values of the 4-bit shift register.

```
enable: 0
clear_n: 0
shift_load: 0
data_bus: (0, 0, 0, 0)
data_serial: 0
clock: 0
output_bus: (0, 0, 0, 0)
output_serial: 0
```

16.13.4 __call__

```
__call__(
    enable=None,
    clear_n=None,
    shift_load=None,
```

(continues on next page)

(continued from previous page)

```
    data_bus=None,
    data_serial=None,
    clock=None,
    output_bus=None,
    output_serial=None
)
```

Force specific values on the wires of the 4-bit shift register.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

16.14 ShiftRegister8

16.14.1 Class `bw.state.ShiftRegister8`

Defined in bitwise/state/SHIFT.py.

8-bit shift register.

16.14.2 __init__

```
__init__(
    enable,
    clear_n,
    shift_load,
    data_bus,
    data_serial,
    clock,
    output_bus,
    output_serial
)
```

Construct a new 8-bit shift register.

Args:

- `enable`: An object of type `Wire`. Enables the shift register.
- `clear_n`: An object of type `Wire`. Clears `output_bus` and `output_serial` to 0 asynchronously if its value is 0.
- `shift_load`: An object of type `Wire`. The mode select. A value of 0 indicates a parallel load operation, where `output_bus` takes on the value of `data_bus`. A value of 1 indicates a shift-right operation, where `output_bus[7]` takes on the value of `output_bus[6]`, `output_bus[6]` takes on the value of `output_bus[5]`, and so on; `output_bus[0]` takes on the value of `data_serial`.
- `data_bus`: An object of type `Bus8`. The parallel data input.
- `data_serial`. An object of type `Wire`. The serial data input.
- `clock`. An object of type `Wire` or `Clock`. The clock input to the shift register.
- `output_bus`. An object of type `Bus8`. The parallel data output.

- `output_serial`. An object of type `Wire`. The serial data output. Identical to `output_bus[7]`.

Raises:

- `TypeError`: If either `data_bus` or `output_bus` is not a bus of width 8.

16.14.3 __str__

Print out the wire values of the 8-bit shift register.

```
enable: 0
clear_n: 0
shift_load: 0
data_bus: (0, 0, 0, 0, 0, 0, 0, 0)
data_serial: 0
clock: 0
output_bus: (0, 0, 0, 0, 0, 0, 0, 0)
output_serial: 0
```

16.14.4 __call__

```
__call__(
    enable=None,
    clear_n=None,
    shift_load=None,
    data_bus=None,
    data_serial=None,
    clock=None,
    output_bus=None,
    output_serial=None
)
```

Force specific values on the wires of the 8-bit shift register.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

16.15 ShiftRegister16

16.15.1 Class `bw.state.ShiftRegister16`

Defined in bitwise/state/SHIFT.py.

16-bit shift register.

16.15.2 __init__

```
__init__(
    enable,
    clear_n,
    shift_load,
    data_bus,
    data_serial,
    clock,
    output_bus,
    output_serial
)
```

Construct a new 16-bit shift register.

Args:

- `enable`: An object of type `Wire`. Enables the shift register.
- `clear_n`: An object of type `Wire`. Clears `output_bus` and `output_serial` to 0 asynchronously if its value is 0.
- `shift_load`: An object of type `Wire`. The mode select. A value of 0 indicates a parallel load operation, where `output_bus` takes on the value of `data_bus`. A value of 1 indicates a shift-right operation, where `output_bus[15]` takes on the value of `output_bus[14]`, `output_bus[14]` takes on the value of `output_bus[13]`, and so on; `output_bus[0]` takes on the value of `data_serial`.
- `data_bus`: An object of type `Bus16`. The parallel data input.
- `data_serial`. An object of type `Wire`. The serial data input.
- `clock`. An object of type `Wire` or `Clock`. The clock input to the shift register.
- `output_bus`. An object of type `Bus16`. The parallel data output.
- `output_serial`. An object of type `Wire`. The serial data output. Identical to `output_bus[15]`.

Raises:

- `TypeError`: If either `data_bus` or `output_bus` is not a bus of width 16.

16.15.3 __str__

Print out the wire values of the 16-bit shift register.

```
enable: 0
clear_n: 0
shift_load: 0
data_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
data_serial: 0
clock: 0
output_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
output_serial: 0
```

16.15.4 __call__

```
__call__(
    enable=None,
    clear_n=None,
    shift_load=None,
    data_bus=None,
    data_serial=None,
    clock=None,
    output_bus=None,
    output_serial=None
)
```

Force specific values on the wires of the 16-bit shift register.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

16.16 UpCounterMod4

16.16.1 Class `bw.state.UpCounterMod4`

Defined in bitwise/state/COUNT.py.

2-bit (mod-4) up counter with asynchronous clear.

16.16.2 __init__

```
__init__(
    enable,
    clear_n,
    clock,
    output_1,
    output_2
)
```

Construct a new mod-4 up counter.

Args:

- `enable`: An object of type `Wire`. Enables the counter.
- `clear_n`: An object of type `Wire`. Clears `output_1` and `output_2` to 0 asynchronously if its value is 0.
- `clock`: An object of type `Wire` or `Clock`. The clock input to the counter.
- `output_1`: An object of type `Wire`. The most significant bit of the output.
- `output_2`: An object of type `Wire`. The least significant bit of the output.

16.16.3 __str__

Print out the wire values of the mod-4 up counter.

```
enable: 0
clear_n: 0
clock: 0
output_1: 0
output_2: 0
```

16.16.4 __call__

```
__call__(
    enable=None,
    clear_n=None,
    clock=None,
    output_1=None,
    output_2=None
)
```

Force specific values on the wires of the mod-4 up counter.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

16.17 UpCounterMod8

16.17.1 Class `bw.state.UpCounterMod8`

Defined in bitwise/state/COUNT.py.

3-bit (mod-8) up counter with asynchronous clear.

16.17.2 __init__

```
__init__(
    enable,
    clear_n,
    clock,
    output_1,
    output_2,
    output_3
)
```

Construct a new mod-8 up counter.

Args:

- `enable`: An object of type `Wire`. Enables the counter.

- `clear_n`: An object of type `Wire`. Clears `output_1`, `output_2`, and `output_3` to 0 asynchronously if its value is 0.
- `clock`: An object of type `Wire` or `Clock`. The clock input to the counter.
- `output_1`: An object of type `Wire`. The most significant bit of the output.
- `output_2`: An object of type `Wire`.
- `output_3`: An object of type `Wire`. The least significant bit of the output.

16.17.3 __str__

Print out the wire values of the mod-8 up counter.

```
enable: 0
clear_n: 0
clock: 0
output_1: 0
output_2: 0
output_3: 0
```

16.17.4 __call__

```
__call__(
    enable=None,
    clear_n=None,
    clock=None,
    output_1=None,
    output_2=None,
    output_3=None
)
```

Force specific values on the wires of the mod-8 up counter.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

16.18 UpCounterMod16

16.18.1 Class `bw.state.UpCounterMod16`

Defined in bitwise/state/COUNT.py.

4-bit (mod-16) up counter with asynchronous clear.

16.18.2 __init__

```
__init__(
    enable,
    clear_n,
    clock,
```

(continues on next page)

(continued from previous page)

```
    output_bus
)
```

Construct a new mod-16 up counter.

Args:

- `enable`: An object of type `Wire`. Enables the counter.
- `clear_n`: An object of type `Wire`. Clears `output_bus` to 0 asynchronously if its value is 0.
- `clock`: An object of type `Wire` or `` `Clock ``. The clock input to the counter.
- `output_bus`: An object of type `Bus4`. The output of the counter. `output_bus[0]` and `output_bus[3]` are the most and least significant bit, respectively.

Raises:

- `TypeError`: If `output_bus` is not a bus of width 4.

16.18.3 __str__

Print out the wire values of the mod-16 up counter.

```
enable: 0
clear_n: 0
clock: 0
output_bus: (0, 0, 0, 0)
```

16.18.4 __call__

```
__call__(
    enable=None,
    clear_n=None,
    clock=None,
    output_bus=None
)
```

Force specific values on the wires of the mod-16 up counter.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

CHAPTER 17

Storage

17.1 DFlipFlop

17.1.1 Class `bw.storage.DFlipFlop`

Defined in bitwise/storage/FLOP.py.

Positive edge-triggered D flip-flop.

17.1.2 __init__

```
__init__(
    data,
    clock,
    output,
    output_not
)
```

Construct a new positive edge-triggered D flip-flop.

Args:

- `data`: An object of type `Wire`. The data input to the flip-flop.
- `clock`: An object of type `Wire` or `Clock`. The clock input to the flip-flop.
- `output`: An object of type `Wire`. The output of the flip-flop. Takes on the value of `data` on the positive edges of `clock`.
- `output_not`: An object of type `Wire`. The complemented form of `output`.

17.1.3 __str__

Print out the wire values of the D flip-flop.

```
data: 0
clock: 0
output: 0
output_not: 0
```

17.1.4 __call__

```
__call__(
    data=None,
    clock=None,
    output=None,
    output_not=None
)
```

Force specific values on the wires of the D flip-flop.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

17.2 DFlipFlopPresetClear

17.2.1 Class `bw.storage.DFlipFlopPresetClear`

Defined in bitwise/storage/FLOP.py.

Positive edge-triggered D flip-flop with asynchronous active low preset and clear.

17.2.2 __init__

```
__init__(
    data,
    preset_n,
    clear_n,
    clock,
    output,
    output_not
)
```

Construct a new positive edge-triggered D flip-flop with preset/clear capabilities.

Args:

- `data`: An object of type `Wire`. The data input to the flip-flop.
- `preset_n`: An object of type `Wire`. Presets `output` to 1 and `output_not` to 0 asynchronously if its value is 0.

- `clear_n`: An object of type `Wire`. Clears `output` to 0 and `output_not` to 1 asynchronously if its value is 0.
- `clock`: An object of type `Wire` or `Clock`. The clock input to the flip-flop.
- `output`: An object of type `Wire`. The output of the flip-flop. Takes on the value of `data` on the positive edges of `clock`.
- `output_not`: An object of type `Wire`. The complemented form of `output`.

17.2.3 __str__

Print out the wire values of the D flip-flop with preset/clear capabilities.

```
data: 0
preset_n: 0
clear_n: 0
clock: 0
output: 0
output_not: 0
```

17.2.4 __call__

```
__call__(
    data=None,
    preset_n=None,
    clear_n=None,
    clock=None,
    output=None,
    output_not=None
)
```

Force specific values on the wires of the D flip-flop with preset/clear capabilities.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

17.3 GatedDLatch

17.3.1 Class `bw.storage.GatedDLatch`

Defined in bitwise/storage/FLOP.py.

Gated D latch.

17.3.2 __init__

```
__init__(
    data,
    clock,
    output,
```

(continues on next page)

(continued from previous page)

```
    output_not
)
```

Construct a new gated D latch.

Args:

- `data`: An object of type `Wire`. The data input to the latch.
- `clock`: An object of type `Wire` or `Clock`. The clock input to the latch.
- `output`: An object of type `Wire`. The output of the latch. Takes on the value of `data` if the value of `clock` is 1.
- `output_not`: An object of type `Wire`. The complemented form of `output`.

17.3.3 __str__

Print out the wire values of the gated D latch.

```
data: 0
clock: 0
output: 0
output_not: 0
```

17.3.4 __call__

```
__call__(
    data=None,
    clock=None,
    output=None,
    output_not=None
)
```

Force specific values on the wires of the gated D latch.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

17.4 GatedSRLatch

17.4.1 Class `bw.storage.GatedSRLatch`

Defined in bitwise/storage/FLOP.py.

Gated SR latch.

17.4.2 __init__

```
__init__(
    set,
    reset,
    clock,
    output,
    output_not
)
```

Construct a new gated SR latch.

Args:

- `set`: An object of type `Wire`. The set input to the latch.
- `reset`: An object of type `Wire`. The reset input to the latch.
- `clock`: An object of type `Wire` or `Clock`. The clock input to the latch.
- `output`: An object of type `Wire`. The output of the latch. When the value of `clock` is 1, takes on the value of 1 if the value of `set` is 1 and the value of 0 if the value of `reset` is 1.
- `output_not`: An object of type `Wire`. The complemented form of `output`.

17.4.3 __str__

Print out the wire values of the gated SR latch.

```
set: 0
reset: 0
clock: 0
output: 0
output_not: 0
```

17.4.4 __call__

```
__call__(
    set=None,
    reset=None,
    clock=None,
    output=None,
    output_not=None
)
```

Force specific values on the wires of the gated SR latch.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

17.5 JKFlipFlop

17.5.1 Class `bw.storage.JKFlipFlop`

Defined in bitwise/storage/FLOP.py.

Positive edge-triggered JK flip-flop.

17.5.2 __init__

```
__init__(
    J,
    K,
    clock,
    output,
    output_not
)
```

Construct a new positive edge-triggered JK flip-flop.

Args:

- `J`: An object of type `Wire`. The J input to the flip-flop.
- `K`: An object of type `Wire`. The K input to the flip-flop.
- `clock`: An object of type `Wire` or `Clock`. The clock input to the flip-flop.
- `output`: An object of type `Wire`. The output of the flip-flop. On the positive edges of `clock`, takes on the value of 1 if the value of `J` is 1, takes on the value of 0 if the value of `K` is 1, and toggles its value if both `J` and `K` have value 1.
- `output_not`: An object of type `Wire`. The complemented form of `output`.

17.5.3 __str__

Print out the wire values of the JK flip-flop.

```
J: 0
K: 0
clock: 0
output: 0
output_not: 0
```

17.5.4 __call__

```
__call__(
    J=None,
    K=None,
    clock=None,
```

(continues on next page)

(continued from previous page)

```
    output=None,
    output_not=None
)
```

Force specific values on the wires of the JK flip-flop.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

17.6 JKFlipFlopPresetClear

17.6.1 Class `bw.storage.JKFlipFlopPresetClear`

Defined in bitwise/storage/FLOP.py.

Positive edge-triggered JK flip-flop with asynchronous active low preset and clear.

17.6.2 __init__

```
__init__(
    J,
    K,
    preset_n,
    clear_n,
    clock,
    output,
    output_not
)
```

Construct a new positive edge-triggered JK flip-flop with preset/clear capabilities.

Args:

- `J`: An object of type `Wire`. The J input to the flip-flop.
- `K`: An object of type `Wire`. The K input to the flip-flop.
- `preset_n`: An object of type `Wire`. Presets `output` to 1 and `output_not` to 0 asynchronously if its value is 0.
- `clear_n`: An object of type `Wire`. Clears `output` to 0 and `output_not` to 1 asynchronously if its value is 0.
- `clock`: An object of type `Wire` or `Clock`. The clock input to the flip-flop.
- `output`: An object of type `Wire`. The output of the flip-flop. On the positive edges of `clock`, takes on the value of 1 if the value of `J` is 1, takes on the value of 0 if the value of `K` is 1, and toggles its value if both `J` and `K` have value 1.
- `output_not`: An object of type `Wire`. The complemented form of `output`.

17.6.3 __str__

Print out the wire values of the JK flip-flop with preset/clear capabilities.

```
J: 0
K: 0
preset_n: 0
clear_n: 0
clock: 0
output: 0
output_not: 0
```

17.6.4 __call__

```
__call__(
    J=None,
    K=None,
    preset_n=None,
    clear_n=None,
    clock=None,
    output=None,
    output_not=None
)
```

Force specific values on the wires of the JK flip-flop with preset/clear capabilities.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

17.7 RAM16x4

17.7.1 Class `bw.storage.RAM16x4`

Defined in bitwise/storage/RAM.py.

16-word deep 4-bit wide random access memory.

17.7.2 __init__

```
__init__(
    data_bus,
    address_bus,
    write_enable,
    clock,
    output_bus
)
```

Construct a new 16-word deep 4-bit wide random access memory array.

Args:

- `data_bus`: An object of type `Bus4`. The data input in write operations.
- `address_bus`: An object of type `Bus4`. The address from which data is read from and written to.
- `write_enable`: An object of type `Wire`. The write enable input. A value of 1 indicates a write operation, while a value of 0 indicates a read-only operation (the value on data_bus is ignored).
- `clock`: An object of type `Wire` or `Clock`. The clock input.
- `output_bus`: An object of type `Bus4`. The currently stored data in the at the address indicated by `address_bus`.

Raises:

- `TypeError`: If either `data_bus`, `address_bus`, or `output_bus` is not a bus of width 4.

17.7.3 __str__

Print out the wire values of the random access memory array.

```
data_bus: (0, 0, 0, 0)
address_bus: (0, 0, 0, 0)
write_enable: 0
clock: 0
output_bus: (0, 0, 0, 0)
```

17.7.4 __call__

```
__call__(
    data_bus=None,
    address_bus=None,
    write_enable=None,
    clock=None,
    output_bus=None
)
```

Force specific values on the wires of the random access memory array.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

17.8 RAM256x4

17.8.1 Class `bw.storage.RAM256x4`

Defined in bitwise/storage/RAM.py.

256-word deep 4-bit wide random access memory.

17.8.2 __init__

```
__init__(
    data_bus,
    address_bus,
    write_enable,
    clock,
    output_bus
)
```

Construct a new 256-word deep 4-bit wide random access memory array.

Args:

- `data_bus`: An object of type `Bus4`. The data input in write operations.
- `address_bus`: An object of type `Bus8`. The address from which data is read from and written to.
- `write_enable`: An object of type `Wire`. The write enable input. A value of 1 indicates a write operation, while a value of 0 indicates a read-only operation (the value on data_bus is ignored).
- `clock`: An object of type `Wire` or `Clock`. The clock input.
- `output_bus`: An object of type `Bus4`. The currently stored data in the at the address indicated by `address_bus`.

Raises:

- `TypeError`: If either `data_bus` or `output_bus` is not a bus of width 4, or if `address_bus` is not a bus of width 8.

17.8.3 __str__

Print out the wire values of the random access memory array.

```
data_bus: (0, 0, 0, 0)
address_bus: (0, 0, 0, 0, 0, 0, 0, 0)
write_enable: 0
clock: 0
output_bus: (0, 0, 0, 0)
```

17.8.4 __call__

```
__call__(
    data_bus=None,
    address_bus=None,
    write_enable=None,
    clock=None,
    output_bus=None
)
```

Force specific values on the wires of the random access memory array.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

17.9 RAM65536x4

17.9.1 Class `bw.storage.RAM65536x4`

Defined in bitwise/storage/RAM.py.

65536-word deep 4-bit wide random access memory.

17.9.2 __init__

```
__init__(
    data_bus,
    address_bus,
    write_enable,
    clock,
    output_bus
)
```

Construct a new 65536-word deep 4-bit wide random access memory array.

Args:

- `data_bus`: An object of type `Bus4`. The data input in write operations.
- `address_bus`: An object of type `Bus16`. The address from which data is read from and written to.
- `write_enable`: An object of type `Wire`. The write enable input. A value of 1 indicates a write operation, while a value of 0 indicates a read-only operation (the value on data_bus is ignored).
- `clock`: An object of type `Wire` or `Clock`. The clock input.
- `output_bus`: An object of type `Bus4`. The currently stored data in the at the address indicated by `address_bus`.

Raises:

- `TypeError`: If either `data_bus` or `output_bus` is not a bus of width 4, or if `address_bus` is not a bus of width 16.

17.9.3 __str__

Print out the wire values of the random access memory array.

```
data_bus: (0, 0, 0, 0)
address_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
write_enable: 0
clock: 0
output_bus: (0, 0, 0, 0)
```

17.9.4 __call__

```
__call__(
    data_bus=None,
    address_bus=None,
    write_enable=None,
    clock=None,
    output_bus=None
)
```

Force specific values on the wires of the random access memory array.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

17.10 RAM16x8

17.10.1 Class `bw.storage.RAM16x8`

Defined in bitwise/storage/RAM.py.

16-word deep 8-bit wide random access memory.

17.10.2 __init__

```
__init__(
    data_bus,
    address_bus,
    write_enable,
    clock,
    output_bus
)
```

Construct a new 16-word deep 8-bit wide random access memory array.

Args:

- `data_bus`: An object of type `Bus8`. The data input in write operations.
- `address_bus`: An object of type `Bus4`. The address from which data is read from and written to.
- `write_enable`: An object of type `Wire`. The write enable input. A value of 1 indicates a write operation, while a value of 0 indicates a read-only operation (the value on data_bus is ignored).
- `clock`: An object of type `Wire` or `Clock`. The clock input.
- `output_bus`: An object of type `Bus8`. The currently stored data in the at the address indicated by `address_bus`.

Raises:

- `TypeError`: If either `data_bus` or `output_bus` is not a bus of width 8, or if `address_bus` is not a bus of width 4.

17.10.3 __str__

Print out the wire values of the random access memory array.

```
data_bus: (0, 0, 0, 0, 0, 0, 0, 0)
address_bus: (0, 0, 0, 0)
write_enable: 0
clock: 0
output_bus: (0, 0, 0, 0, 0, 0, 0, 0)
```

17.10.4 __call__

```
__call__(
    data_bus=None,
    address_bus=None,
    write_enable=None,
    clock=None,
    output_bus=None
)
```

Force specific values on the wires of the random access memory array.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

17.11 RAM256x8

17.11.1 Class `bw.storage.RAM256x8`

Defined in bitwise/storage/RAM.py.

256-word deep 8-bit wide random access memory.

17.11.2 __init__

```
__init__(
    data_bus,
    address_bus,
    write_enable,
    clock,
    output_bus
)
```

Construct a new 256-word deep 8-bit wide random access memory array.

Args:

- `data_bus`: An object of type `Bus8`. The data input in write operations.
- `address_bus`: An object of type `Bus8`. The address from which data is read from and written to.

- `write_enable`: An object of type `Wire`. The write enable input. A value of 1 indicates a write operation, while a value of 0 indicates a read-only operation (the value on data_bus is ignored).
- `clock`: An object of type `Wire` or `Clock`. The clock input.
- `output_bus`: An object of type `Bus8`. The currently stored data in the at the address indicated by `address_bus`.

Raises:

- `TypeError`: If either `data_bus`, `address_bus`, or `output_bus` is not a bus of width 8.

17.11.3 __str__

Print out the wire values of the random access memory array.

```
data_bus: (0, 0, 0, 0, 0, 0, 0, 0)
address_bus: (0, 0, 0, 0, 0, 0, 0, 0)
write_enable: 0
clock: 0
output_bus: (0, 0, 0, 0, 0, 0, 0, 0)
```

17.11.4 __call__

```
__call__(
    data_bus=None,
    address_bus=None,
    write_enable=None,
    clock=None,
    output_bus=None
)
```

Force specific values on the wires of the random access memory array.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

17.12 RAM65536x8

17.12.1 Class `bw.storage.RAM65536x8`

Defined in bitwise/storage/RAM.py.

65536-word deep 8-bit wide random access memory.

17.12.2 __init__

```
__init__(
    data_bus,
    address_bus,
    write_enable,
    clock,
    output_bus
)
```

Construct a new 65536-word deep 8-bit wide random access memory array.

Args:

- `data_bus`: An object of type `Bus8`. The data input in write operations.
- `address_bus`: An object of type `Bus16`. The address from which data is read from and written to.
- `write_enable`: An object of type `Wire`. The write enable input. A value of 1 indicates a write operation, while a value of 0 indicates a read-only operation (the value on data_bus is ignored).
- `clock`: An object of type `Wire` or `Clock`. The clock input.
- `output_bus`: An object of type `Bus8`. The currently stored data in the at the address indicated by `address_bus`.

Raises:

- `TypeError`: If either `data_bus` or `output_bus` is not a bus of width 8, or if `address_bus` is not a bus of width 16.

17.12.3 __str__

Print out the wire values of the random access memory array.

```
data_bus: (0, 0, 0, 0, 0, 0, 0, 0)
address_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
write_enable: 0
clock: 0
output_bus: (0, 0, 0, 0, 0, 0, 0, 0)
```

17.12.4 __call__

```
__call__(
    data_bus=None,
    address_bus=None,
    write_enable=None,
    clock=None,
    output_bus=None
)
```

Force specific values on the wires of the random access memory array.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

17.13 RAM16x16

17.13.1 Class `bw.storage.RAM16x16`

Defined in bitwise/storage/RAM.py.

16-word deep 16-bit wide random access memory.

17.13.2 __init__

```
__init__(
    data_bus,
    address_bus,
    write_enable,
    clock,
    output_bus
)
```

Construct a new 16-word deep 16-bit wide random access memory array.

Args:

- `data_bus`: An object of type `Bus16`. The data input in write operations.
- `address_bus`: An object of type `Bus4`. The address from which data is read from and written to.
- `write_enable`: An object of type `Wire`. The write enable input. A value of 1 indicates a write operation, while a value of 0 indicates a read-only operation (the value on data_bus is ignored).
- `clock`: An object of type `Wire` or `Clock`. The clock input.
- `output_bus`: An object of type `Bus16`. The currently stored data in the at the address indicated by `address_bus`.

Raises:

- `TypeError`: If either `data_bus` or `output_bus` is not a bus of width 16, or if `address_bus` is not a bus of width 4.

17.13.3 __str__

Print out the wire values of the random access memory array.

```
data_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
address_bus: (0, 0, 0, 0)
write_enable: 0
clock: 0
output_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
```

17.13.4 __call__

```
__call__(
    data_bus=None,
    address_bus=None,
    write_enable=None,
    clock=None,
    output_bus=None
)
```

Force specific values on the wires of the random access memory array.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

17.14 RAM256x16

17.14.1 Class `bw.storage.RAM256x16`

Defined in bitwise/storage/RAM.py.

256-word deep 16-bit wide random access memory.

17.14.2 __init__

```
__init__(
    data_bus,
    address_bus,
    write_enable,
    clock,
    output_bus
)
```

Construct a new 256-word deep 16-bit wide random access memory array.

Args:

- `data_bus`: An object of type `Bus16`. The data input in write operations.
- `address_bus`: An object of type `Bus8`. The address from which data is read from and written to.
- `write_enable`: An object of type `Wire`. The write enable input. A value of 1 indicates a write operation, while a value of 0 indicates a read-only operation (the value on data_bus is ignored).
- `clock`: An object of type `Wire` or `Clock`. The clock input.
- `output_bus`: An object of type `Bus16`. The currently stored data in the at the address indicated by `address_bus`.

Raises:

- `TypeError`: If either `data_bus` or `output_bus` is not a bus of width 16, or if `address_bus` is not a bus of width 8.

17.14.3 __str__

Print out the wire values of the random access memory array.

```
data_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
address_bus: (0, 0, 0, 0, 0, 0, 0, 0)
write_enable: 0
clock: 0
output_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
```

17.14.4 __call__

```
__call__(
    data_bus=None,
    address_bus=None,
    write_enable=None,
    clock=None,
    output_bus=None
)
```

Force specific values on the wires of the random access memory array.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

17.15 RAM65536x16

17.15.1 Class `bw.storage.RAM65536x16`

Defined in bitwise/storage/RAM.py.

65536-word deep 16-bit wide random access memory.

17.15.2 __init__

```
__init__(
    data_bus,
    address_bus,
    write_enable,
    clock,
    output_bus
)
```

Construct a new 65536-word deep 16-bit wide random access memory array.

Args:

- `data_bus`: An object of type `Bus16`. The data input in write operations.
- `address_bus`: An object of type `Bus16`. The address from which data is read from and written to.

- `write_enable`: An object of type `Wire`. The write enable input. A value of 1 indicates a write operation, while a value of 0 indicates a read-only operation (the value on data_bus is ignored).
- `clock`: An object of type `Wire` or `Clock`. The clock input.
- `output_bus`: An object of type `Bus16`. The currently stored data in the at the address indicated by `address_bus`.

Raises:

- `TypeError`: If either `data_bus`, `address_bus`, or `output_bus` is not a bus of width 16.

17.15.3 __str__

Print out the wire values of the random access memory array.

```
data_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
address_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
write_enable: 0
clock: 0
output_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
```

17.15.4 __call__

```
__call__(
    data_bus=None,
    address_bus=None,
    write_enable=None,
    clock=None,
    output_bus=None
)
```

Force specific values on the wires of the random access memory array.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

17.16 Register4

17.16.1 Class `bw.storage.Register4`

Defined in bitwise/storage/REG.py.

4-bit storage register.

17.16.2 __init__

```
__init__(
    data_bus,
    enable,
    clock,
    output_bus
)
```

Construct a new 4-bit storage register.

Args:

- `data_bus`: An object of type `Bus4`. The data input to the register.
- `enable`: An object of type `Wire`. Enables the register.
- `clock`: An object of type `Wire` or `Clock`. The clock input to the register.
- `output_bus`: An object of type `Bus4`. The output of the register. Takes on the value of `data_bus` on the positive edges of `clock` if the value of `enable` is 1.

Raises:

- `TypeError`: If either `data_bus` or `output_bus` is not a bus of width 4.

17.16.3 __str__

Print out the wire values of the 4-bit storage register.

```
data_bus: (0, 0, 0, 0)
enable: 0
clock: 0
output_bus: (0, 0, 0, 0)
```

17.16.4 __call__

```
__call__(
    data_bus=None,
    enable=None,
    clock=None,
    output_bus=None
)
```

Force specific values on the wires of the 4-bit storage register.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

17.17 Register8

17.17.1 Class `bw.storage.Register8`

Defined in bitwise/storage/REG.py.

8-bit storage register.

17.17.2 __init__

```
__init__(
    data_bus,
    enable,
    clock,
    output_bus
)
```

Construct a new 8-bit storage register.

Args:

- `data_bus`: An object of type `Bus8`. The data input to the register.
- `enable`: An object of type `Wire`. Enables the register.
- `clock`: An object of type `Wire` or `Clock`. The clock input to the register.
- `output_bus`: An object of type `Bus8`. The output of the register. Takes on the value of `data_bus` on the positive edges of `clock` if the value of `enable` is 1.

Raises:

- `TypeError`: If either `data_bus` or `output_bus` is not a bus of width 8.

17.17.3 __str__

Print out the wire values of the 8-bit storage register.

```
data_bus: (0, 0, 0, 0, 0, 0, 0, 0)
enable: 0
clock: 0
output_bus: (0, 0, 0, 0, 0, 0, 0, 0)
```

17.17.4 __call__

```
__call__(
    data_bus=None,
    enable=None,
    clock=None,
    output_bus=None
)
```

Force specific values on the wires of the 8-bit storage register.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

17.18 Register16

17.18.1 Class `bw.storage.Register16`

Defined in bitwise/storage/REG.py.

16-bit storage register.

17.18.2 __init__

```
__init__(
    data_bus,
    enable,
    clock,
    output_bus
)
```

Construct a new 16-bit storage register.

Args:

- `data_bus`: An object of type `Bus16`. The data input to the register.
- `enable`: An object of type `Wire`. Enables the register.
- `clock`: An object of type `Wire` or `Clock`. The clock input to the register.
- `output_bus`: An object of type `Bus16`. The output of the register. Takes on the value of `data_bus` on the positive edges of `clock` if the value of `enable` is 1.

Raises:

- `TypeError`: If either `data_bus` or `output_bus` is not a bus of width 16.

17.18.3 __str__

Print out the wire values of the 16-bit storage register.

```
data_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
enable: 0
clock: 0
output_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
```

17.18.4 __call__

```
__call__(
    data_bus=None,
    enable=None,
    clock=None,
```

(continues on next page)

(continued from previous page)

```
    output_bus=None
)
```

Force specific values on the wires of the 16-bit storage register.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

17.19 SRLatch

17.19.1 Class `bw.storage.SRLatch`

Defined in bitwise/storage/FLOP.py.

SR latch.

17.19.2 __init__

```
__init__(
    set,
    reset,
    output,
    output_not
)
```

Construct a new SR latch.

Args:

- `set`: An object of type `Wire`. The set input to the latch.
- `reset`: An object of type `Wire`. The reset input to the latch.
- `output`: An object of type `Wire`. The output of the latch. Takes on the value of 1 if the value of `set` is 1 and the value of 0 if the value of `reset` is 1.
- `output_not`: An object of type `Wire`. The complemented form of `output`.

17.19.3 __str__

Print out the wire values of the SR latch.

```
set: 0
reset: 0
output: 0
output_not: 0
```

17.19.4 __call__

```
__call__(
    set=None,
    reset=None,
    output=None,
    output_not=None
)
```

Force specific values on the wires of the SR latch.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

17.20 TFlipFlop

17.20.1 Class `bw.storage.TFlipFlop`

Defined in bitwise/storage/FLOP.py.

Positive edge-triggered T flip-flop.

17.20.2 __init__

```
__init__(
    toggle,
    clock,
    output,
    output_not
)
```

Construct a new positive edge-triggered T flip-flop.

Args:

- `toggle`: An object of type `Wire`. The toggle input to the flip-flop.
- `clock`: An object of type `Wire` or `Clock`. The clock input to the flip-flop.
- `output`: An object of type `Wire`. The output of the flip-flop. Toggles its value on the positive edges of `clock` if the value of `toggle` is 1.
- `output_not`: An object of type `Wire`. The complemented form of `output`.

17.20.3 __str__

Print out the wire values of the T flip-flop.

```
toggle: 0
clock: 0
output: 0
output_not: 0
```

17.20.4 __call__

```
__call__(
    toggle=None,
    clock=None,
    output=None,
    output_not=None
)
```

Force specific values on the wires of the T flip-flop.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

17.21 TFlipFlopPresetClear

17.21.1 Class bw.storage.TFlipFlopPresetClear

Defined in bitwise/storage/FLOP.py.

Positive edge-triggered T flip-flop with asynchronous active low preset and clear.

17.21.2 __init__

```
__init__(
    toggle,
    preset_n,
    clear_n,
    clock,
    output,
    output_not
)
```

Construct a new positive edge-triggered T flip-flop with preset/clear capabilities.

Args:

- `toggle`: An object of type `Wire`. The toggle input to the flip-flop.
- `preset_n`: An object of type `Wire`. Presets `output` to 1 and `output_not` to 0 asynchronously if its value is 0.
- `clear_n`: An object of type `Wire`. Clears `output` to 0 and `output_not` to 1 asynchronously if its value is 0.
- `clock`: An object of type `Wire` or `Clock`. The clock input to the flip-flop.
- `output`: An object of type `Wire`. The output of the flip-flop. Toggles its value on the positive edges of `clock` if the value of `toggle` is 1.
- `output_not`: An object of type `Wire`. The complemented form of `output`.

17.21.3 __str__

Print out the wire values of the T flip-flop with preset/clear capabilities.

```
toggle: 0
preset_n: 0
clear_n: 0
clock: 0
output: 0
output_not: 0
```

17.21.4 __call__

```
__call__(
    toggle=None,
    preset_n=None,
    clear_n=None,
    clock=None,
    output=None,
    output_not=None
)
```

Force specific values on the wires of the T flip-flop with preset/clear capabilities.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

CHAPTER 18

Wire

18.1 BufferBus4

18.1.1 Class `bw.wire.BufferBus4`

Defined in bitwise/wire/TRI_BUS.py.

4-bit buffered bus.

18.1.2 __init__

```
__init__(
    enable,
    input_bus,
    output_bus
)
```

Initialize a new tri-state buffer with buses of width 4 as input and output.

Args:

- `enable`: An object of type `Wire`.
- `input_bus`: An object of type `Bus4`.
- `output_bus`: An object of type `Bus4`. Takes on the value of `input_bus` if `enable` has value 1. Otherwise, value is independent of `input_bus`.

Raises:

- `TypeError`: If either `input_bus` or `output_bus` is not a bus of width 4.

18.1.3 __str__

Print out the wire values of the 4-bit-wide tri-state buffer.

```
enable: 0
input_bus: (0, 0, 0, 0)
output_bus: (0, 0, 0, 0)
```

18.1.4 __call__

```
__call__(
    enable=None,
    input_bus=None,
    output_bus=None
)
```

Force specific values on the wires of the 4-bit-wide tri-state buffer.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

18.2 BufferBus8

18.2.1 Class `bw.wire.BufferBus8`

Defined in bitwise/wire/TRI_BUS.py.

8-bit buffered bus.

18.2.2 __init__

```
__init__(
    enable,
    input_bus,
    output_bus
)
```

Initialize a new tri-state buffer with buses of width 8 as input and output.

Args:

- `enable`: An object of type `Wire`.
- `input_bus`: An object of type `Bus8`.
- `output_bus`: An object of type `Bus8`. Takes on the value of `input_bus` if `enable` has value 1. Otherwise, value is independent of `input_bus`.

Raises:

- `TypeError`: If either `input_bus` or `output_bus` is not a bus of width 8.

18.2.3 __str__

Print out the wire values of the 8-bit-wide tri-state buffer.

```
enable: 0
input_bus: (0, 0, 0, 0, 0, 0, 0, 0)
output_bus: (0, 0, 0, 0, 0, 0, 0, 0)
```

18.2.4 __call__

```
__call__(
    enable=None,
    input_bus=None,
    output_bus=None
)
```

Force specific values on the wires of the 8-bit-wide tri-state buffer.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

18.3 BufferBus16

18.3.1 Class `bw.wire.BufferBus16`

Defined in bitwise/wire/TRI_BUS.py.

16-bit buffered bus.

18.3.2 __init__

```
__init__(
    enable,
    input_bus,
    output_bus
)
```

Initialize a new tri-state buffer with buses of width 16 as input and output.

Args:

- `enable`: An object of type `Wire`.
- `input_bus`: An object of type `Bus16`.
- `output_bus`: An object of type `Bus16`. Takes on the value of `input_bus` if `enable` has value 1. Otherwise, value is independent of `input_bus`.

Raises:

- `TypeError`: If either `input_bus` or `output_bus` is not a bus of width 16.

18.3.3 __str__

Print out the wire values of the 16-bit-wide tri-state buffer.

```
enable: 0
input_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
output_bus: (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)
```

18.3.4 __call__

```
__call__(
    enable=None,
    input_bus=None,
    output_bus=None
)
```

Force specific values on the wires of the 16-bit-wide tri-state buffer.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

18.4 Bus4

18.4.1 Class `bw.wire.Bus4`

Defined in bitwise/wire/BUS.py.

4-bit bus.

18.4.2 __init__

```
__init__(
    wire_1,
    wire_2,
    wire_3,
    wire_4
)
```

Initialize a new 4-bit bus.

Args:

- `wire_1`, `wire_2`, ... , `wire_4` (optional): Objects of type `Wire`. If not given, new wires will be created, which can then only be accessed by indexing the bus.

18.4.3 __str__

Print out the wire values of the bus.

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

18.4.4 __call__

```
__call__(
    wire_1=None,
    wire_2=None,
    wire_3=None,
    wire_4=None
)
```

Force specific values on the wires of the bus.

18.4.5 Accessors:

- `wires`: A tuple of the wires in the bus.
- `wire_values`: A tuple of values of the wires in the bus.

18.4.6 Mutators:

- `wire_values`: A tuple of values of the wires in the bus.

18.5 Bus8

18.5.1 Class `bw.wire.Bus8`

Defined in bitwise/wire/BUS.py.

8-bit bus.

18.5.2 __init__

```
__init__(
    wire_1,
    wire_2,
    wire_3,
    wire_4,
    wire_5,
    wire_6,
    wire_7,
    wire_8
)
```

Initialize a new 8-bit bus.

Args:

- `wire_1`, `wire_2`, ... , `wire_8` (optional): Objects of type `Wire`. If not given, new wires will be created, which can then only be accessed by indexing the bus.

18.5.3 __str__

Print out the wire values of the bus.

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

18.5.4 __call__

```
__call__(
    wire_1=None,
    wire_2=None,
    wire_3=None,
    wire_4=None,
    wire_5=None,
    wire_6=None,
    wire_7=None,
    wire_8=None
)
```

Force specific values on the wires of the bus.

18.5.5 Accessors:

- `wires`: A tuple of the wires in the bus.
- `wire_values`: A tuple of values of the wires in the bus.

18.5.6 Mutators:

- `wire_values`: A tuple of values of the wires in the bus.

18.6 Bus16

18.6.1 Class `bw.wire.Bus16`

Defined in bitwise/wire/BUS.py.

16-bit bus.

18.6.2 __init__

```
__init__(
    wire_1,
    wire_2,
    wire_3,
    wire_4,
    wire_5,
    wire_6,
    wire_7,
```

(continues on next page)

(continued from previous page)

```
    wire_8,
    wire_9,
    wire_10,
    wire_11,
    wire_12,
    wire_13,
    wire_14,
    wire_15,
    wire_16
)
```

Initialize a new 16-bit bus.

Args:

- `wire_1`, `wire_2`, ... , `wire_16` (optional): Objects of type `Wire`. If not given, new wires will be created, which can then only be accessed by indexing the bus.

18.6.3 __str__

Print out the wire values of the bus.

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

18.6.4 __call__

```
__call__(
    wire_1=None,
    wire_2=None,
    wire_3=None,
    wire_4=None,
    wire_5=None,
    wire_6=None,
    wire_7=None,
    wire_8=None,
    wire_9=None,
    wire_10=None,
    wire_11=None,
    wire_12=None,
    wire_13=None,
    wire_14=None,
    wire_15=None,
    wire_16=None
)
```

Force specific values on the wires of the bus.

18.6.5 Accessors:

- `wires`: A tuple of the wires in the bus.
- `wire_values`: A tuple of values of the wires in the bus.

18.6.6 Mutators:

- `wire_values`: A tuple of values of the wires in the bus.

18.7 BusSevenSegmentDisplay

18.7.1 Class `bw.wire.BusSevenSegmentDisplay`

Defined in bitwise/wire/BUS.py.

Bus for output to a seven-segment display.

18.7.2 __init__

```
__init__(
    wire_1,
    wire_2,
    wire_3,
    wire_4,
    wire_5,
    wire_6,
    wire_7
)
```

Initialize a new seven-segment display bus.

Args:

- `wire_1`, `wire_2`, ... , `wire_7` (optional): Objects of type `Wire`. If not given, new wires will be created, which can then only be accessed by indexing the bus.

18.7.3 __str__

Print out the wire values of the bus.

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

18.7.4 __call__

```
__call__(
    wire_1=None,
    wire_2=None,
    wire_3=None,
    wire_4=None,
    wire_5=None,
    wire_6=None,
    wire_7=None
)
```

Force specific values on the wires of the bus.

18.7.5 Accessors:

- `wires`: A tuple of the wires in the bus.
- `wire_values`: A tuple of values of the wires in the bus.

18.7.6 Mutators:

- `wire_values`: A tuple of values of the wires in the bus.

18.8 Clock

18.8.1 Class `bw.wire.Clock`

Defined in bitwise/wire/CLK.py.

A clock, with either a 0 or 1 integer value.

This class is identical to the `Wire` class, only differing in semantic purposes.

18.8.2 __init__

```
__init__(
    value=0
)
```

Initialize a new clock with default value 0.

After instantiation, the value of the clock can be both accessed and mutated using `clock.value`. For example:

```
In [1]: import bitwise as bw

In [2]: a = bw.wire.Clock()

In [3]: a.value
Out[3]: 0

In [4]: a.value = 1

In [5]: a.value
Out[5]: 1
```

Raises:

- `ValueError`: If value assigned to clock is not 0 or 1.

18.8.3 __str__

Print out the value of the clock.

```
0
```

18.8.4 __call__

```
__call__(
    value=None
)
```

Force a specific value on the clock.

18.8.5 Accessors:

- `value`: The value of the clock.

18.8.6 Mutators:

- `value`: The value of the clock.

18.9 TristateBuffer

18.9.1 Class `bw.wire.TristateBuffer`

Defined in bitwise/wire/TRI.py.

Tri-state buffer.

18.9.2 __init__

```
__init__(
    enable,
    input,
    output
)
```

Initialize a new tri-state buffer.

Args:

- `enable`: An object of type `Wire`.
- `input`: An object of type `Wire`.
- `output`: An object of type `Wire`. Takes on the value of `input` if `enable` has value 1. Otherwise, value is independent of `input`.

18.9.3 __str__

Print out the wire values of the tri-state buffer.

```
enable: 0
input: 0
output: 0
```

18.9.4 __call__

```
__call__(
    enable=None,
    input_bus=None,
    output_bus=None
)
```

Force specific values on the wires of the tri-state buffer.

Note that this method takes *zero* positional arguments; all values must be given as keyword arguments.

18.10 Wire

18.10.1 Class `bw.wire.Wire`

Defined in bitwise/wire/WIRE.py.

A wire, with either a 0 or 1 integer value.

18.10.2 __init__

```
__init__(
    value=0
)
```

Initialize a new wire with default value 0.

After instantiation, the value of the wire can be both accessed and mutated using `wire.value`. For example:

```
In [1]: import bitwise as bw

In [2]: a = bw.wire.Wire()

In [3]: a.value
Out[3]: 0

In [4]: a.value = 1

In [5]: a.value
Out[5]: 1
```

Raises:

- `ValueError`: If value assigned to wire is not 0 or 1.

18.10.3 __str__

Print out the value of the wire.

```
0
```

18.10.4 __call__

```
__call__(
    value=None
)
```

Force a specific value on the wire.

18.10.5 Accessors:

- `value`: The value of the wire.

18.10.6 Mutators:

- `value`: The value of the wire.

CHAPTER 19

About Bitwise

Bitwise is a Python library intended to make hardware design and simulation more accessible for software engineers. While it can never replace a true hardware description language, it aims to serve as a useful tool in the hardware design process, allowing a user to build and test their digital circuits in a high-level programming language (i.e. with simpler syntax and more flexible semantics) before implementing them using an HDL.

19.1 Quick Example

The following code creates a half-adder circuit:

```
"""
Create a half-adder.
"""
import bitwise as bw

def main():
    # initialize inputs
    a = bw.wire.Wire()
    b = bw.wire.Wire()

    # initialize outputs
    sum_ = bw.wire.Wire()
    carry_out = bw.wire.Wire()

    # create circuit
    bw.gate.XORGate2(a, b, sum_)  # XORs a and b and puts the result into sum_
    bw.gate.ANDGate2(a, b, carry_out)  # ANDs a and b and puts the result into carry_
↪out

if __name__ == "__main__":
    main()
```

Interacting with it in a Python session:

```
In [1]: a.value = 0

In [2]: b.value = 0

In [3]: sum_.value
Out[3]: 0

In [4]: carry_out.value
Out[4]: 0

In [5]: a.value = 1

In [6]: sum_.value
Out[6]: 1

In [7]: carry_out.value
Out[7]: 0

In [8]: b.value = 1

In [9]: sum_.value
Out[9]: 0

In [10]: carry_out.value
Out[10]: 1
```