Amaranth (1, practice)

Hardware System Design

Spring, 2023

Outline

- **Practice**
  - Install Amaranth
  - Implement MAC, Adder tree
- Appendix
  - Amaranth operators
  - Template generator code

Install Amaranth

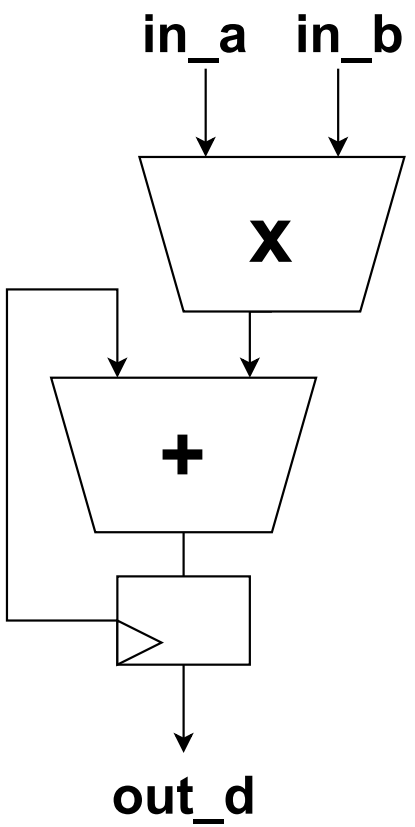
- `pip install --upgrade 'amaranth[builtin-yosys]'`

Installation — Amaranth HDL toolchain 0.4.dev90 documentation (amaranth-lang.org)

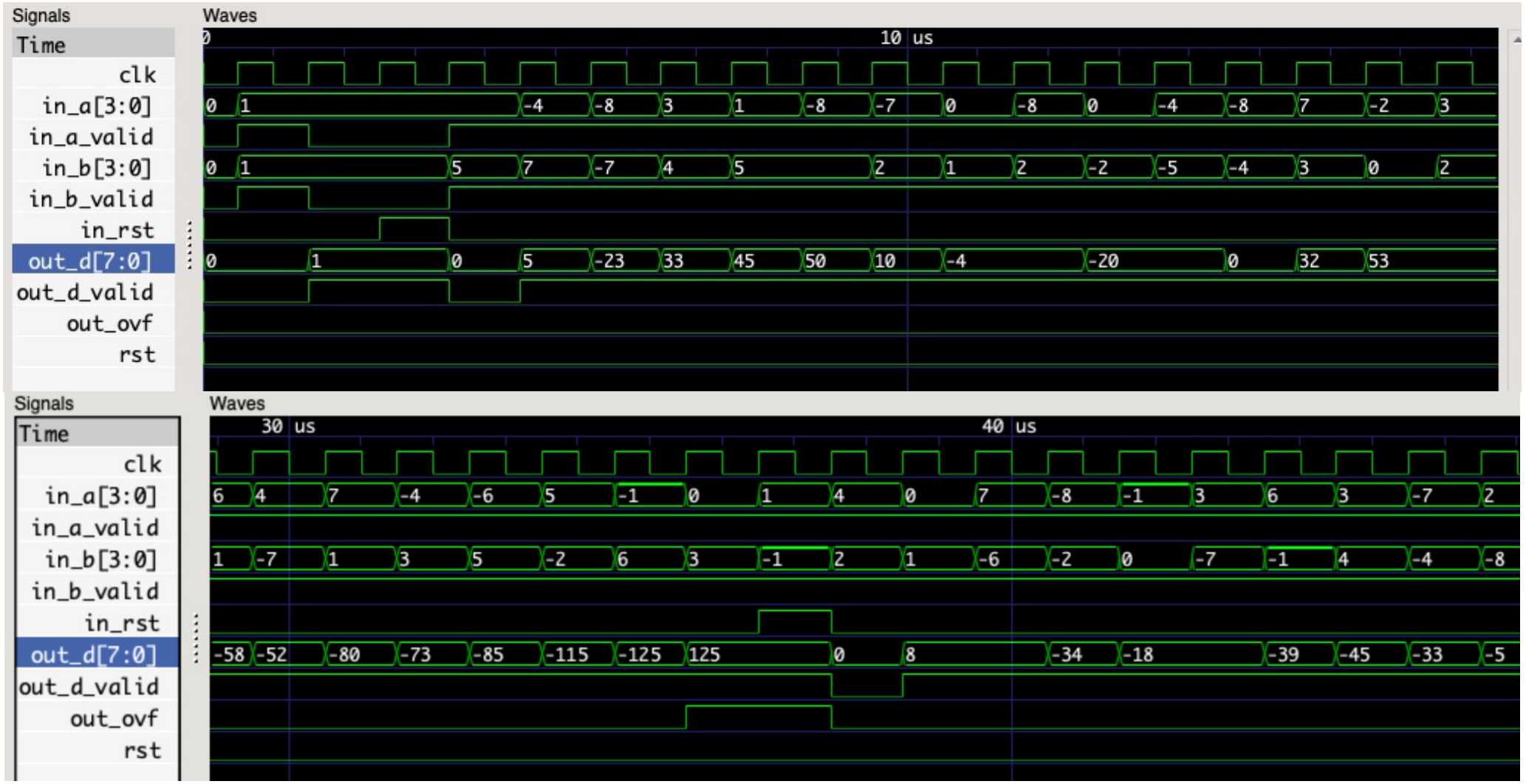
Implement MAC

- MAC := Multiply-Accumulate

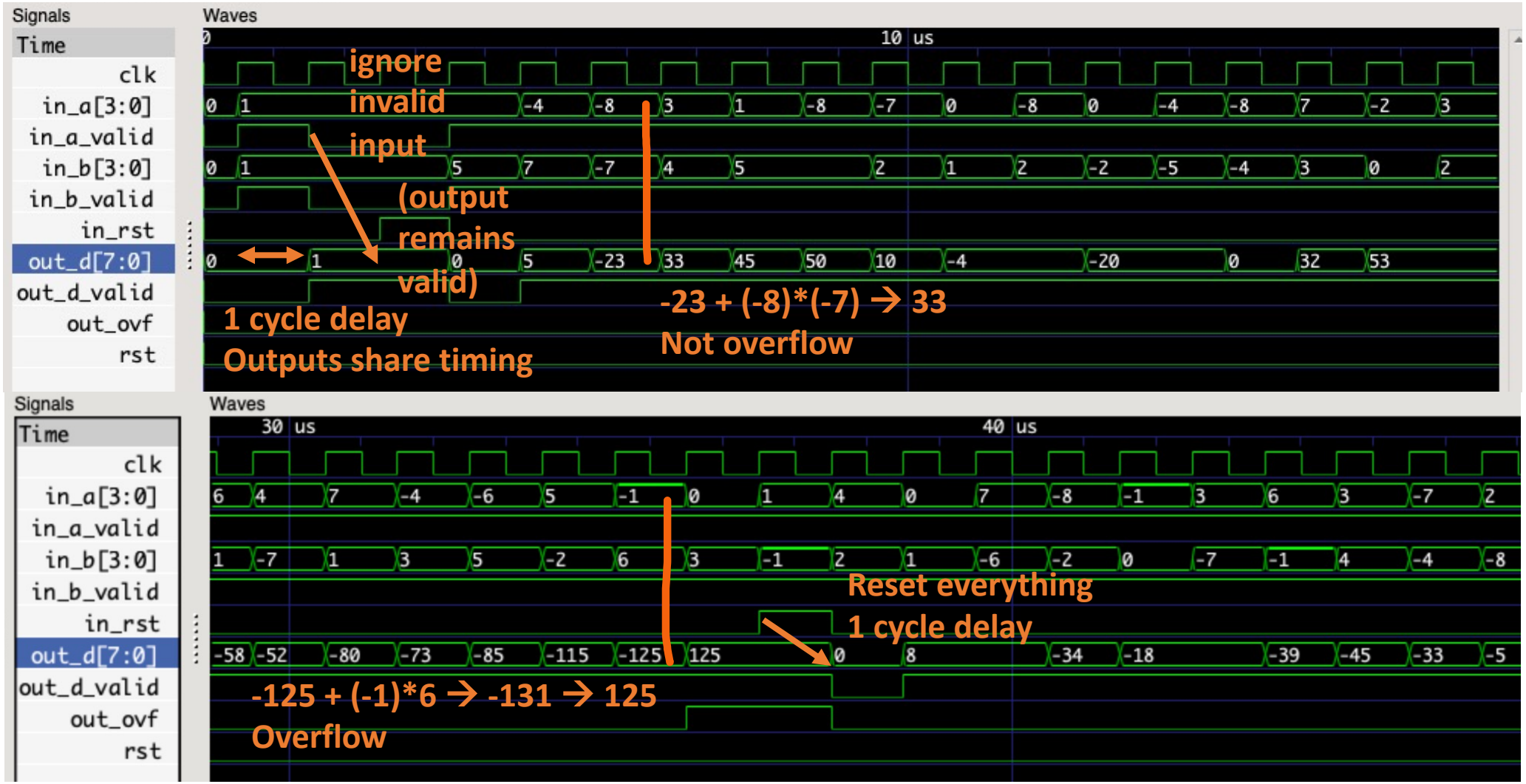
Assume
  combinational adder and multiplier
Should
  support both signed & unsigned input
    consider overflow
    for signed values, overflow iff
      (+) + (+) → (-)
      (-) + (-) → (+)
  `out_d_valid, out_ovf` has
    same timing as `out_d`
  `out_d_valid` is set if
    `in_a_valid & in_b_valid`
  update only if both inputs are valid



Implement MAC



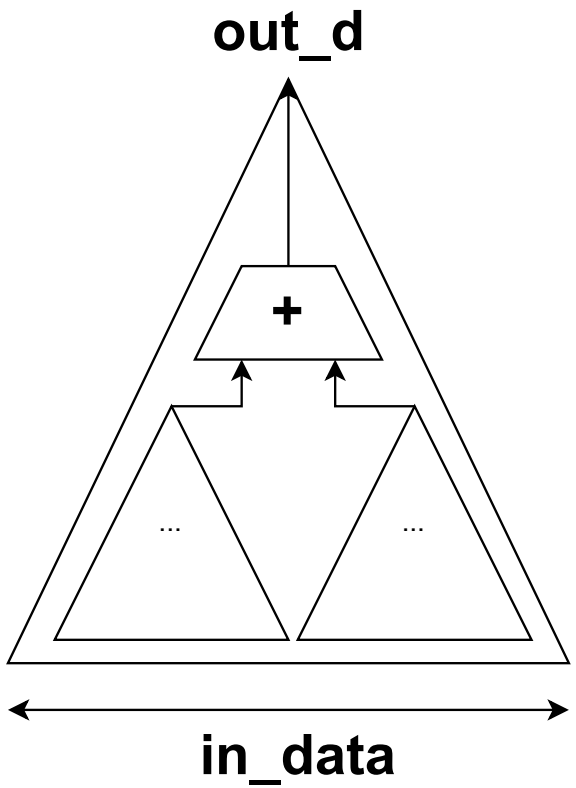
Implement MAC



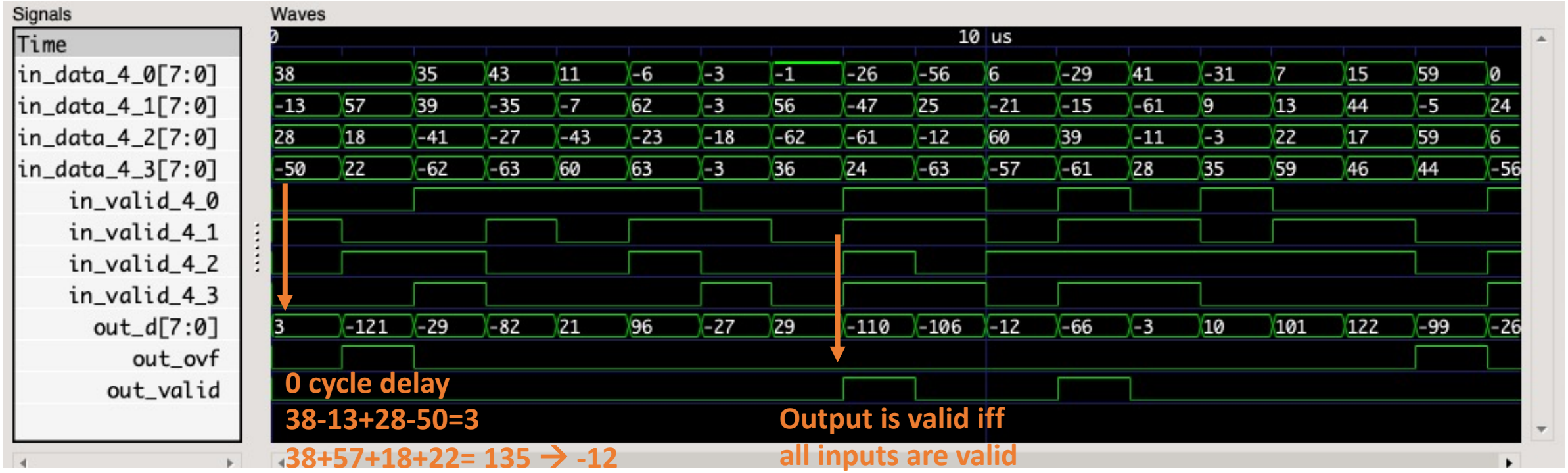
Implement Adder tree

- Recursive adder tree

  combinational logic
  support both signed & unsigned input
  (consider overflow)
  output overflow is set iff
  overflow happened before OR
  during addition
  output is valid iff all inputs are valid



Implement Adder tree



Practice

- Skeleton code
  https://www.notion.so/skeleton-code-d0ee32565e344835b48a529dad4eae12?pvs=4
- Implement and show
  - Waveform
  - Code

Appendix

- Amaranth operators
- Template generator

Amaranth operators

Operation	Description
`a + b`	addition
`-a`	negation
`a - b`	subtraction
`a * b`	multiplication
`a // b`	floor division
`a % b`	modulo
`abs(a)`	absolute value

Operation	Description
`a == b`	equality
`a != b`	inequality
`a < b`	less than
`a <= b`	less than or equal
`a > b`	greater than
`a >= b`	greater than or equal

Operation	Description	Notes
`~a`	bitwise NOT; complement	
`a & b`	bitwise AND	
`a | b`	bitwise OR	
`a ^ b`	bitwise XOR	
`a.implies(b)`	bitwise IMPLY	
`a >> b`	arithmetic right shift by variable amount	1, 2
`a << b`	left shift by variable amount	2
`a.rotate_left(i)`	left rotate by constant amount	3
`a.rotate_right(i)`	right rotate by constant amount	3
`a.shift_left(i)`	left shift by constant amount	3
`a.shift_right(i)`	right shift by constant amount	3

[1] Logical and arithmetic right shift of an unsigned value are equivalent. Logical right shift of a signed value can be expressed by converting it to unsigned first.

[2] *(1, 2)* Shift amount must be unsigned; integer shifts in Python require the amount to be positive.

[3] *(1, 2, 3, 4)* Shift and rotate amounts can be negative, in which case the direction is reversed.

[Language guide(Operators) (amaranth-lang.org)]

Amaranth operators

Operation	Description	Notes
`a.all()`	reduction AND; are all bits set?	4
`a.any()`	reduction OR; is any bit set?	4
`a.xor()`	reduction XOR; is an odd number of bits set?	
`a.bool()`	conversion to boolean; is non-zero?	5

[4] (1, 2) Conceptually the same as applying the Python `all()` or `any()` function to the value viewed as a collection of bits.

[5] Conceptually the same as applying the Python `bool()` function to the value viewed as an integer.

Python expression	Amaranth expression (any operands)
`not a`	`~(a).bool()`
`a and b`	`(a).bool() & (b).bool()`
`a or b`	`(a).bool() | (b).bool()`

Python expression	Amaranth expression (boolean o
`not p`	`~(p)`
`p and q`	`(p) & (q)`
`p or q`	`(p) | (q)`

Amaranth operation	Equivalent Python code
`Cat(a, b)`	`a + b`
`Repl(a, n)`	`a * n`
`a.bit_select(b, w)`	`a[b:b+w]`
`a.word_select(b, w)`	`a[b*w:b*w+w]`

Operation	Description	Notes
`len(a)`	bit length; value width	6
`a[i:j:k]`	bit slicing by constant subscripts	7
`iter(a)`	bit iteration	
`a.bit_select(b, w)`	overlapping part select with variable offset	
`a.word_select(b, w)`	non-overlapping part select with variable offset	
`Cat(a, b)`	concatenation	8
`Repl(a, n)`	replication	

[6] Words "length" and "width" have the same meaning when talking about Amaranth values. Conventionally, "width" is used.

[7] All variations of the Python slice notation are supported, including "extended slicing". E.g. all of `a[0]`, `a[1:9]`, `a[2:]`, `a[:-2]`, `a[::-1]`, `a[0:8:2]` select bits in the same way as other Python sequence types select their elements.

[8] In the concatenated value, `a` occupies the least significant bits, and `b` the most significant bits.

[Language guide(Operators) (amaranth-lang.org)]

Template generator

- https://www.notion.so/Template-generator-20fbe8e38d2e49b19fc9e7eeab3c807f?pvs=4