Meta-Programming

Pat Hanrahan

CS448H: Agile Hardware Design
Winter 2017

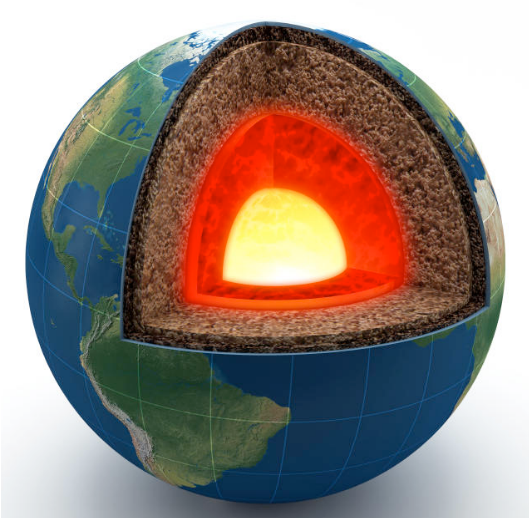
Magma Product Types

```
T = Bit | Array(n,T) | Tuple(T1, T2, …, Tn)
```

- **Recursive product type (not algebraic data type)**
- **All types have fixed size**

```
Array(n,T)

Tuple(T1, T2, …, Tn)
```

- **Calling these functions returns a type (class)**
- **Generalizes *higher-kinded* types**



Mantle

Standard Low-Level Hardware Library
(think libc and libm, TTL 7400 / CMOS 4000)

Mantle
`libc` for hardware

And, Or, Xor, …
Add, Sub, …
Mux,
Registers
Shift registers
Counters
Memories
…

see `mantle.md`

Higher-Order Circuits

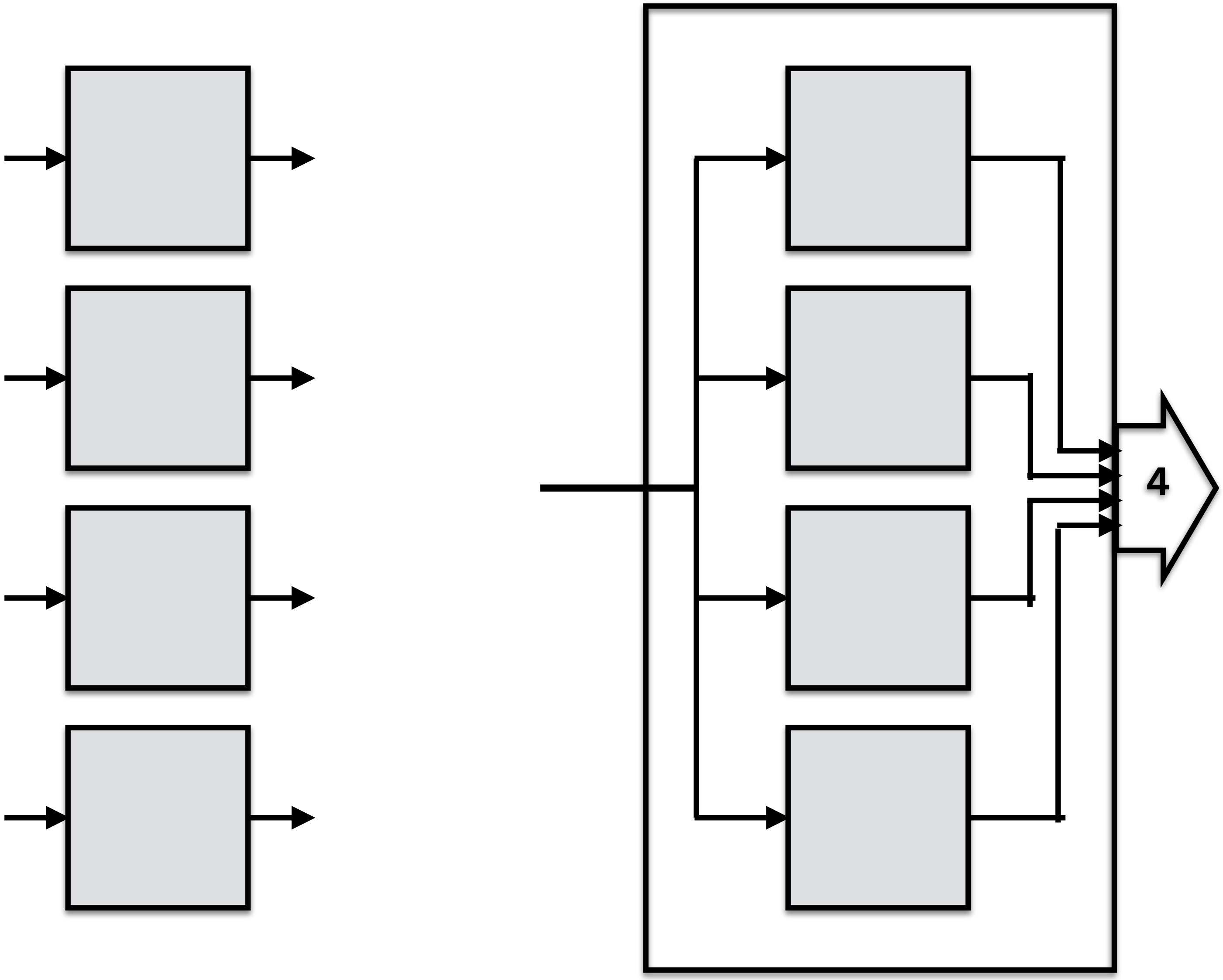
Instantiate Circuits

```
lut4 = LUT4(I0&I1)
dff = DFF()
```

Circuits are "like" functions

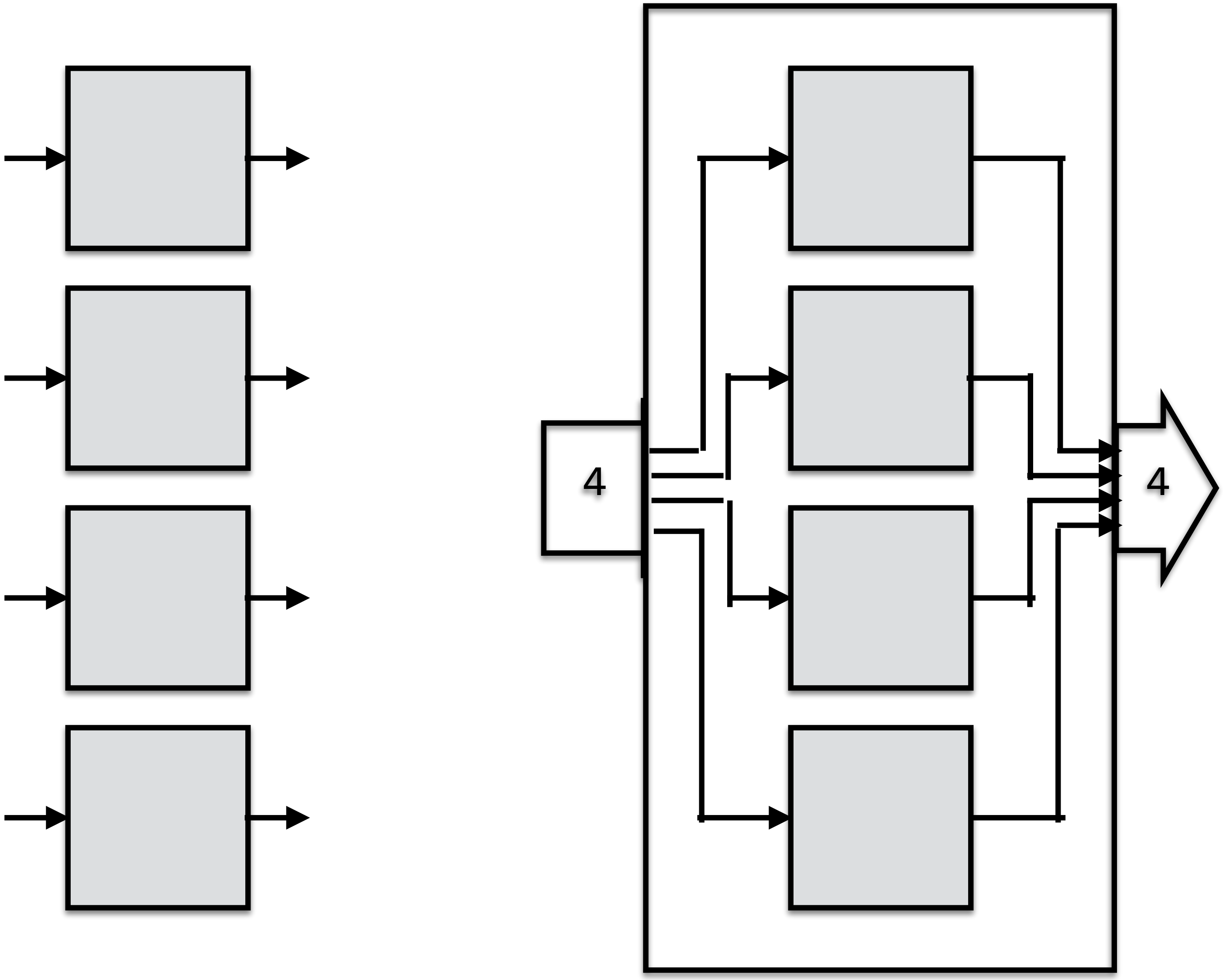
Use higher-order operators to constructor new circuits from other circuits

FullAdder - fulladder/fulladder.py



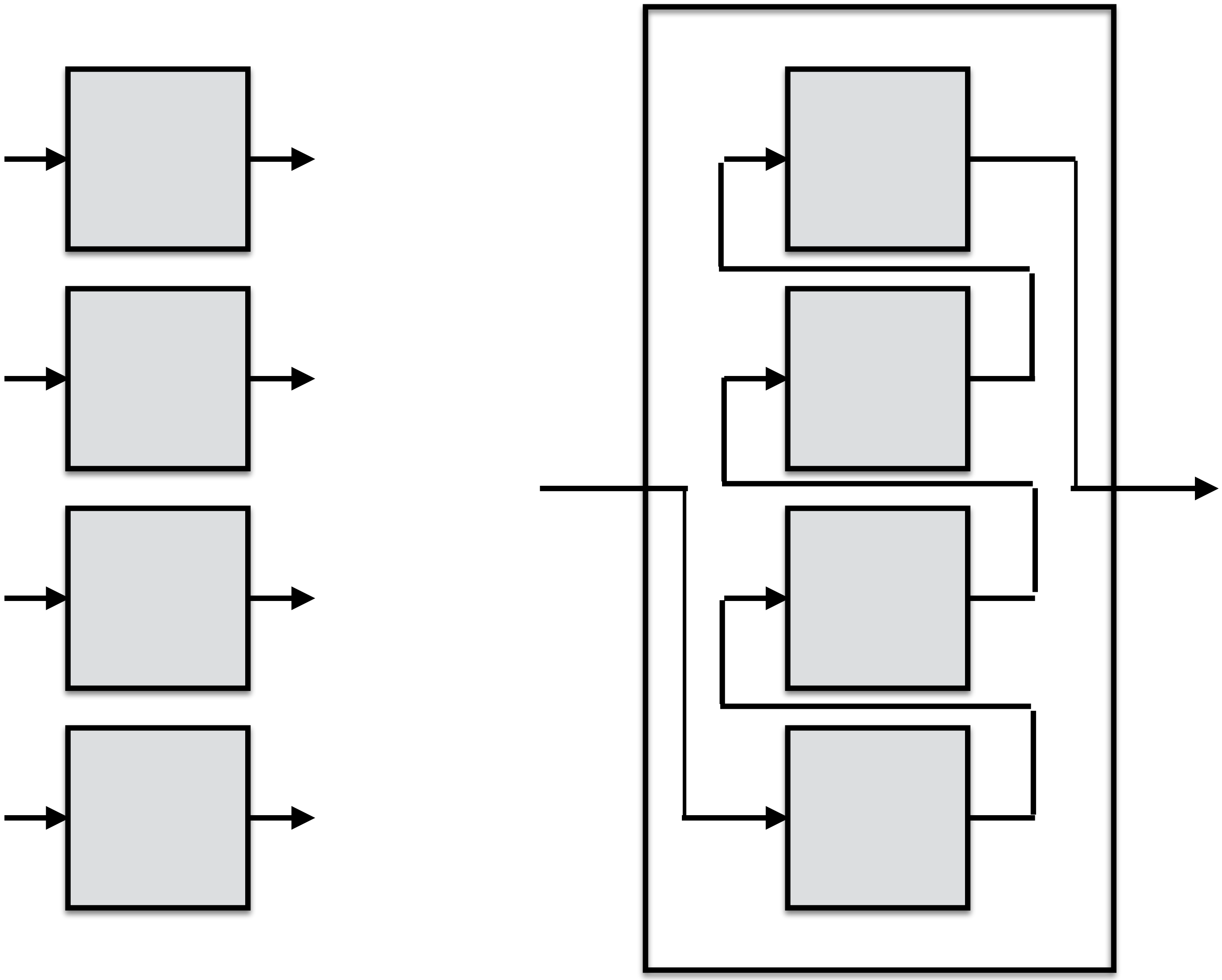
fork

Invert(n)

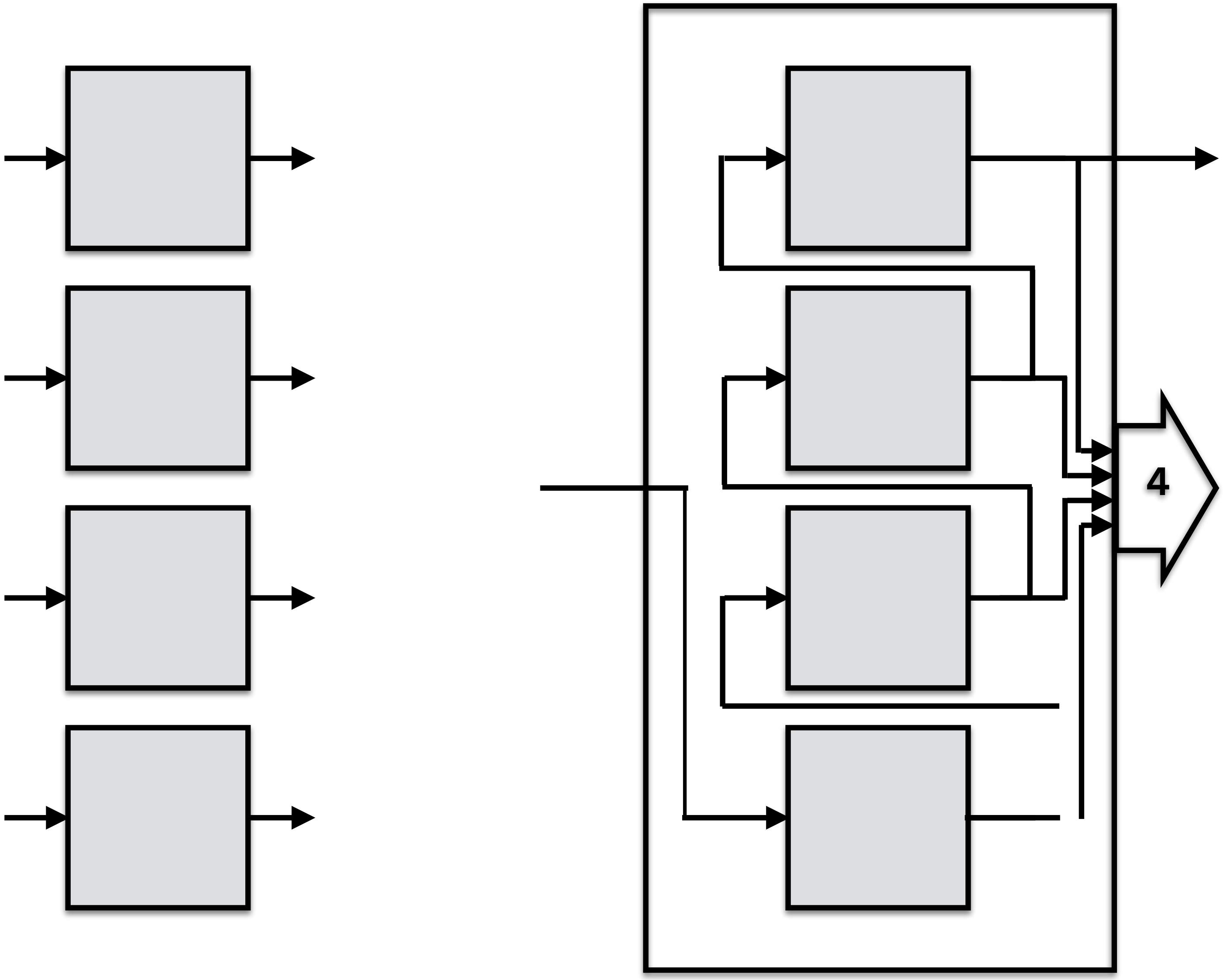


join

Add(n) - add/{add1,add2}.py



fold



scan

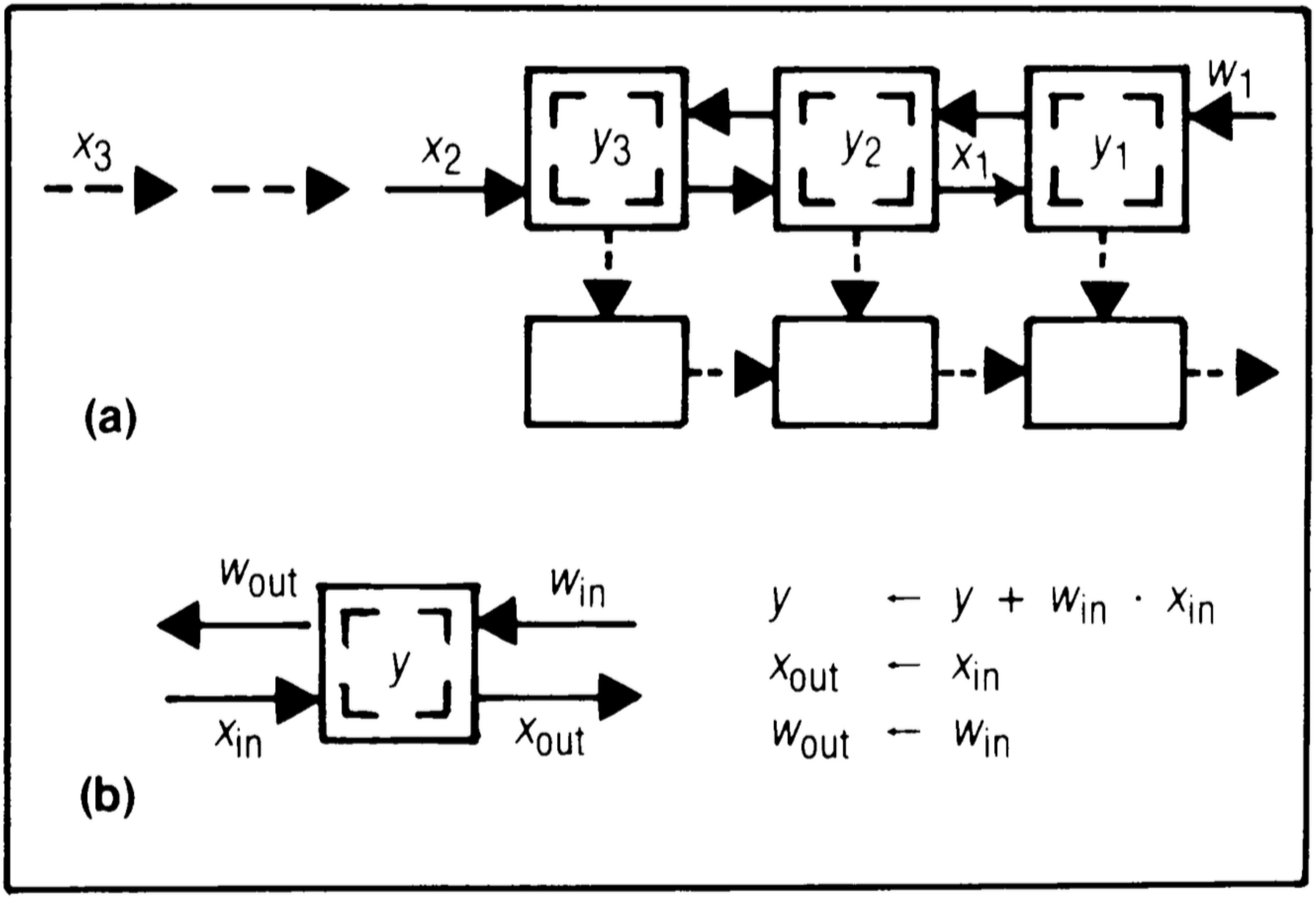

x_3
x_2
y_3
y_2
x_1
y_1
w_1
(a)
w_{out}
w_{in}
y
x_{in}
x_{out}
$y \leftarrow y + w_{in} \cdot x_{in}$
$x_{out} \leftarrow x_{in}$
$w_{out} \leftarrow w_{in}$
(b)

Figure 6. Design R1: systolic convolution array (a) and cell (b) where y_i's stay and x_i's and y_i's move in opposite directions systolically.

```
# Beyond functional programming …

# Higher-order circuits

def braid(circuits,
  joinargs=[],
  flatargs=[],
  forkargs=[],
  foldargs={}, rfoldargs={},
  scanargs={}, rscanargs={}):


…
# easily generalized to 2D
```

```
// uncurry and curry

LUT2 = DeclareCircuit('LUT2',
    "I0", In(Bit), "I1", In(Bit),
     "O", Out(Bit))

lut1 = LUT2()
assert str(lut1.interface) ==
    'I0 : In(Bit), I1 : In(Bit) -> O : Out(Bit)'

lut2 = uncurry(lut1)
assert str(lut2.interface) ==
    'I : Array(2,In(Bit)) -> O : Out(Bit)'

lut3 = curry(lut2)
assert str(lut3.interface) ==
    'I0 : In(Bit), I1 : In(Bit) -> O : Out(Bit)'
```

```
// curry and uncurl

ROM2 = DeclareCircuit('ROM2',
    "I", In(Array(2,Bit)),
    "O", Out(Bit))

rom1 = ROM2()
assert str(rom1.interface) ==
    'I : Array(2,In(Bit)) -> O : Out(Bit)'

rom2 = curry(rom1)
assert str(rom2.interface) ==
    'I0 : In(Bit), I1 : In(Bit) -> O : Out(Bit)'

rom3 = uncurry(rom2)
assert str(rom3.interface) ==
    'I : Array(2,In(Bit)) -> O : Out(Bit)'
```

Circuit Definitions

(Modules)

Circuit Definition

1. **Generate Circuit class (new type)**

```
Register2 = DefineRegister(2)
```

2. **Instance Circuit**

```
register2 = Register2()
```

3. **Wire circuit instances**

```
O = register2(I)
```

```
def DefineRegister(n):
    reg = DefineCircuit('Register'+str(n),
               "I", In(Array(n,Bit)),
               "O", In(Array(n,Bit)),
               "CLK", In(Bit))

    ffs =  join(col(FF, n))
    wire(ffs(reg.I), reg.O)
    wire(reg.CLK, ffs.CLK)

    EndCircuit()
    return reg

Register2 = DefineRegister(2)
register = Register2()
wire(register(I), O)
```

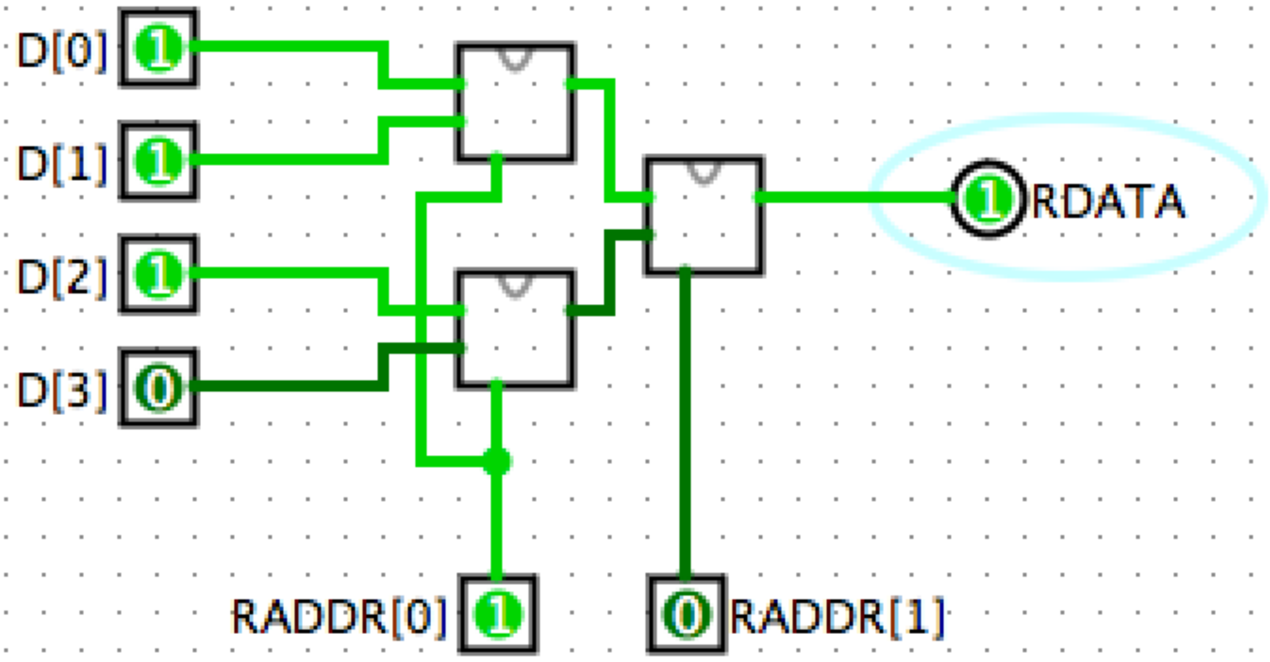
```
def DefineRegister(n):

  T = In(Array(n, Bit))
  class _Register(Circuit):
    name = 'Register'+str(n)
    IO = ["I",T,
          "O",T,
          "CLK", In(Bit)]

    @classmethod
    def definition(reg):
      ffs = join(FFs(n))
      wire(ffs(reg.I), reg.O)
      wire(reg.CLK, ffs.CLK)

    return _Register
```

```
def DefineSISO(n):
  """
    Generate Serial-In, Serial-Out shift register.

    I : Bit -> O : Bit
  """
  class _SISO(Circuit):
    name = 'SISO'+str(n)
    IO = ['input I',Bit, 'output O',Bit]+ClockInterface()

    @classmethod
    def definition(siso):
      ffs = FFs(n)
      reg = braid(ffs, foldargs={"I":"O"})
      reg(siso.I)
      wire(reg.O, siso.O)
      wireclock(siso, reg)

    return _SISO
```

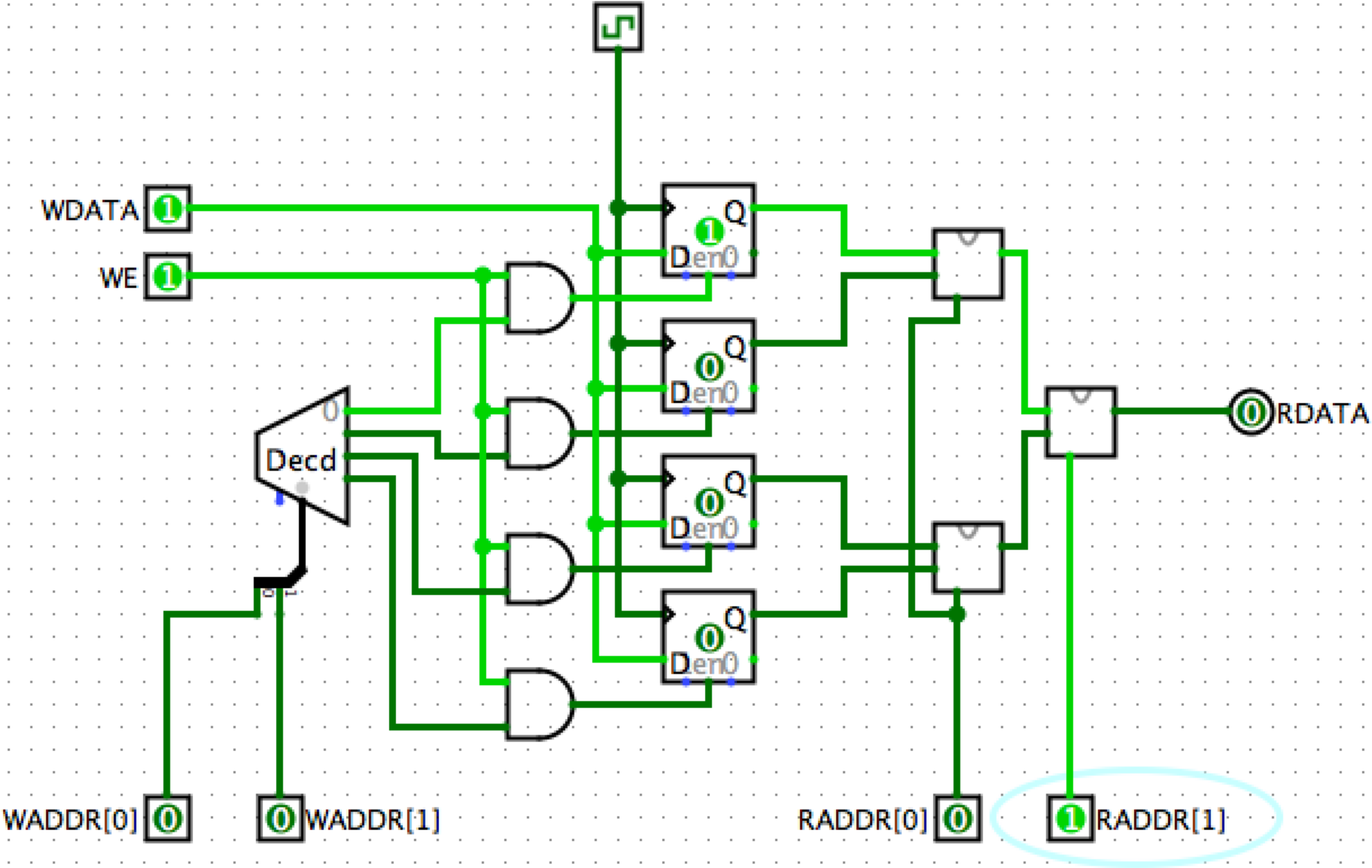
```
def DefineSIPO(n):
  """
    Generate Serial-In, Parallel-Out shift register.

    I : Bit -> O : Array(n, Bit)
  """
  T = Array(n, Bit)
  class _SIPO(Circuit):
    name = 'SIPO'+str(n)
    IO = ['input I',Bit,'output O',T]+ClockInterface()

    @classmethod
    def definition(sipo):
      ffs = FFs(n)
      reg = braid(ffs, scanargs={"I":"O"})
      wire(sipo.I, reg.I)
      wire(reg.O, sipo.O)
      wireclock(sipo, reg)

    return _SIPO
```

ROM and RAM

```
ROM(2) :: RADDR:Array(4,Bit) -> RDATA:Bit
```

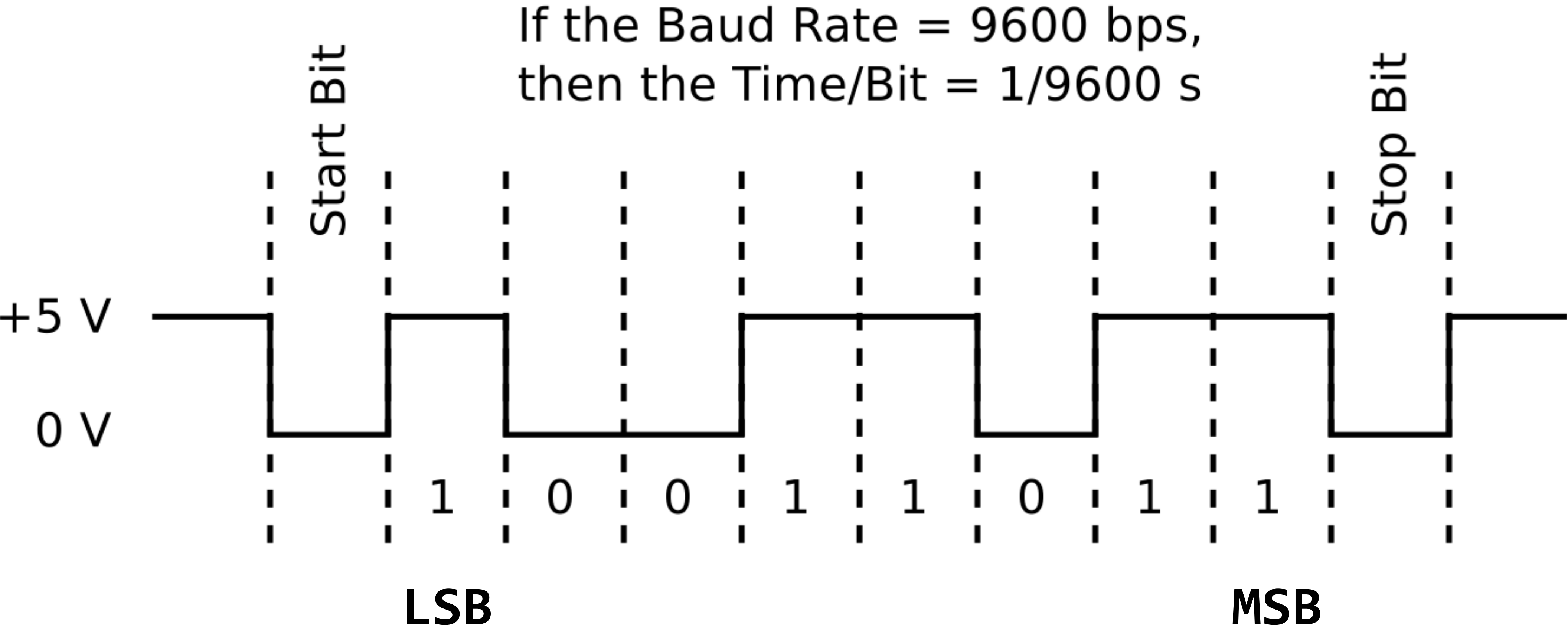


```
RAM(2) :: RADDR:Array(4,Bit),
          WDATA:Bit, WADDR:Array(4,Bit), WE:Bit
       -> RDATA:Bit
```



UART

Serial Protocol

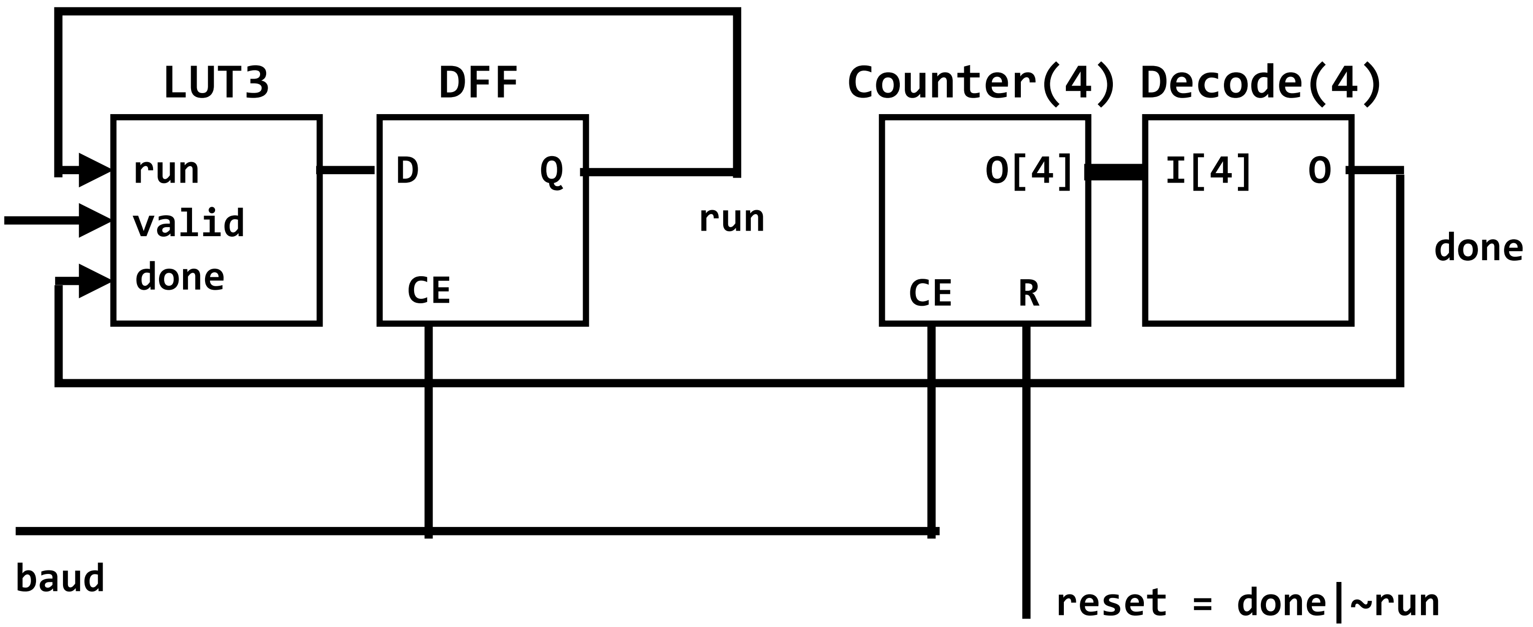


If the Baud Rate = 9600 bps,
then the Time/Bit = 1/9600 s

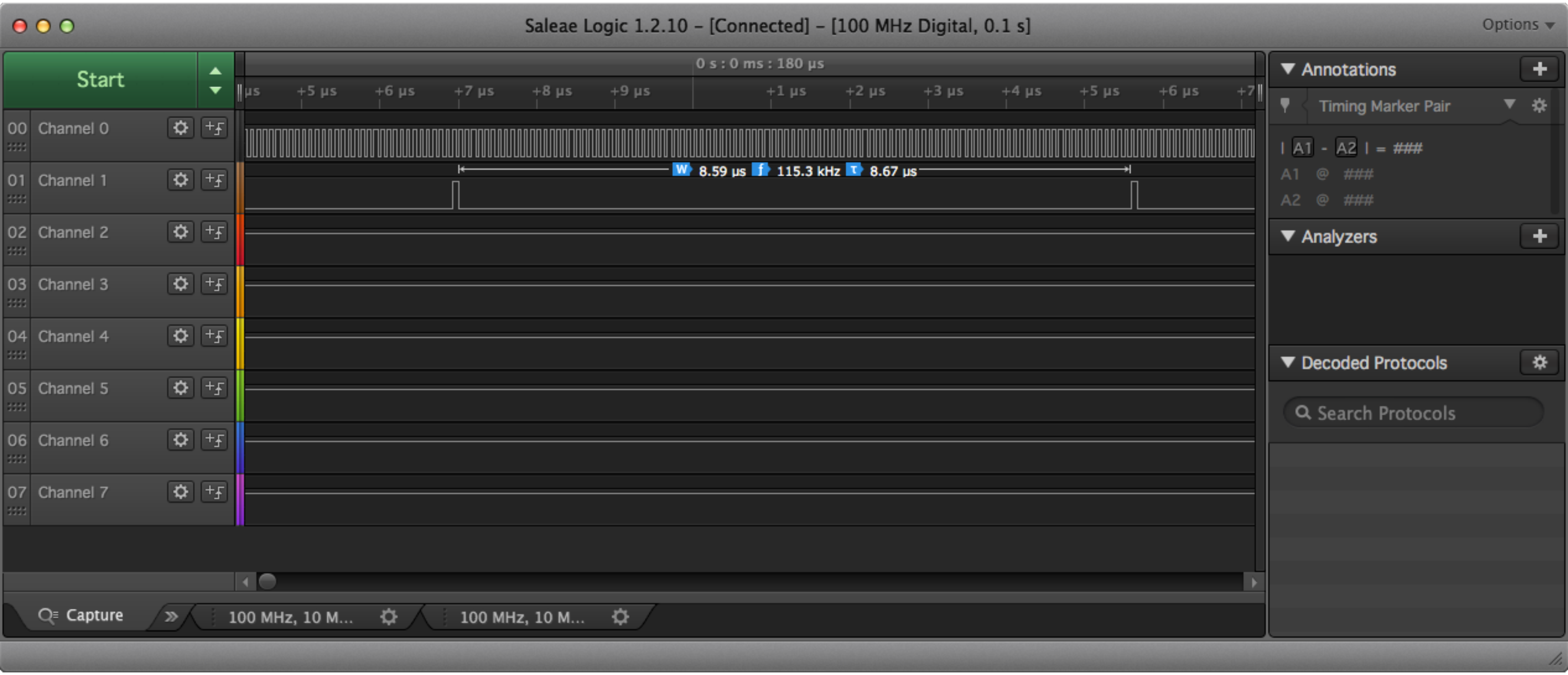
```
void bit(int val)
{
    gpio_write(pin, val);
    delay_us(DELAY);
}

void putc(int c)
{
    bit(0); // start bit
    // output 8-bits, lsb first
    for ( int i = 0; i < 8; i++ ) {
        bit(c & 0x1);
        c >>= 1;
    }
    bit(1); // 2 stop bits
    bit(1);
}
```

fsm.py

RVD	R'
000	0
001	0
010	1
011	0
100	1
101	0
110	1
111	0



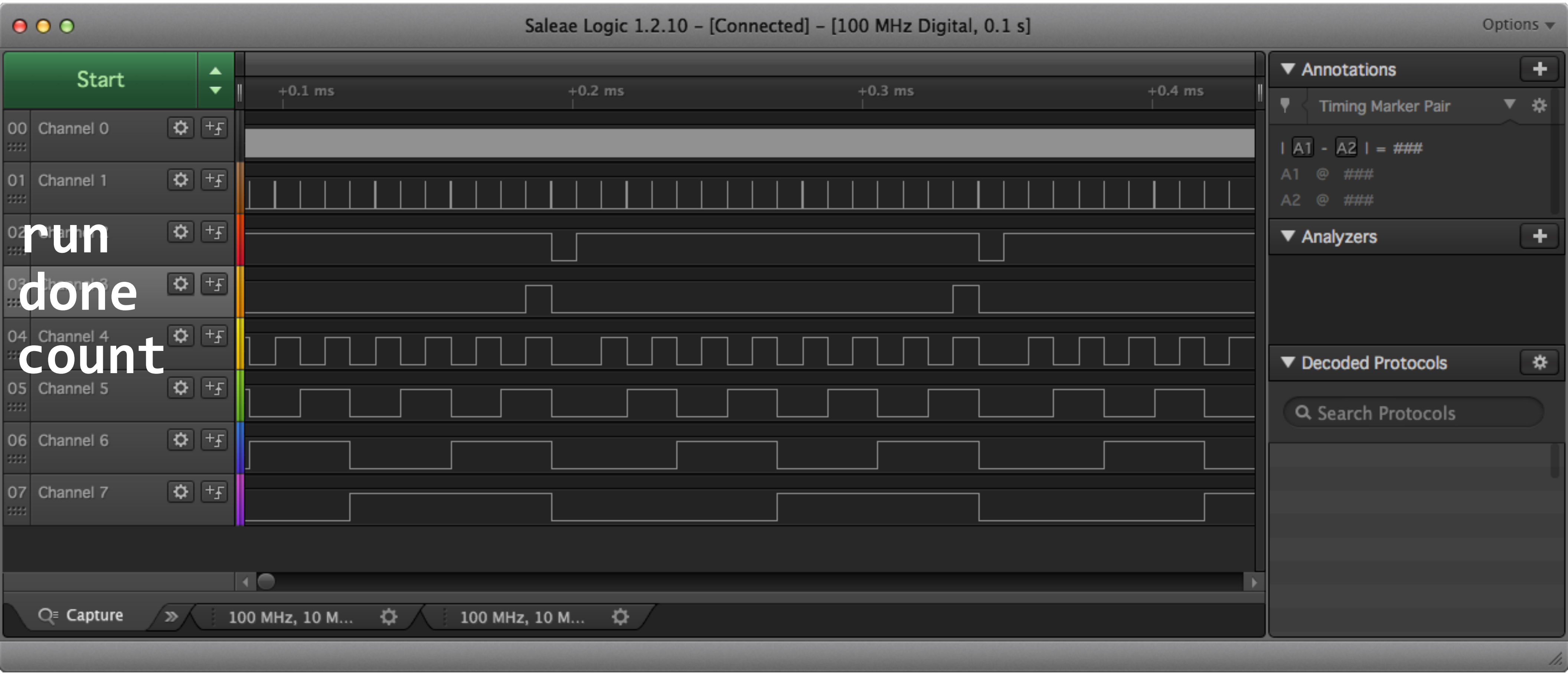
baud.py



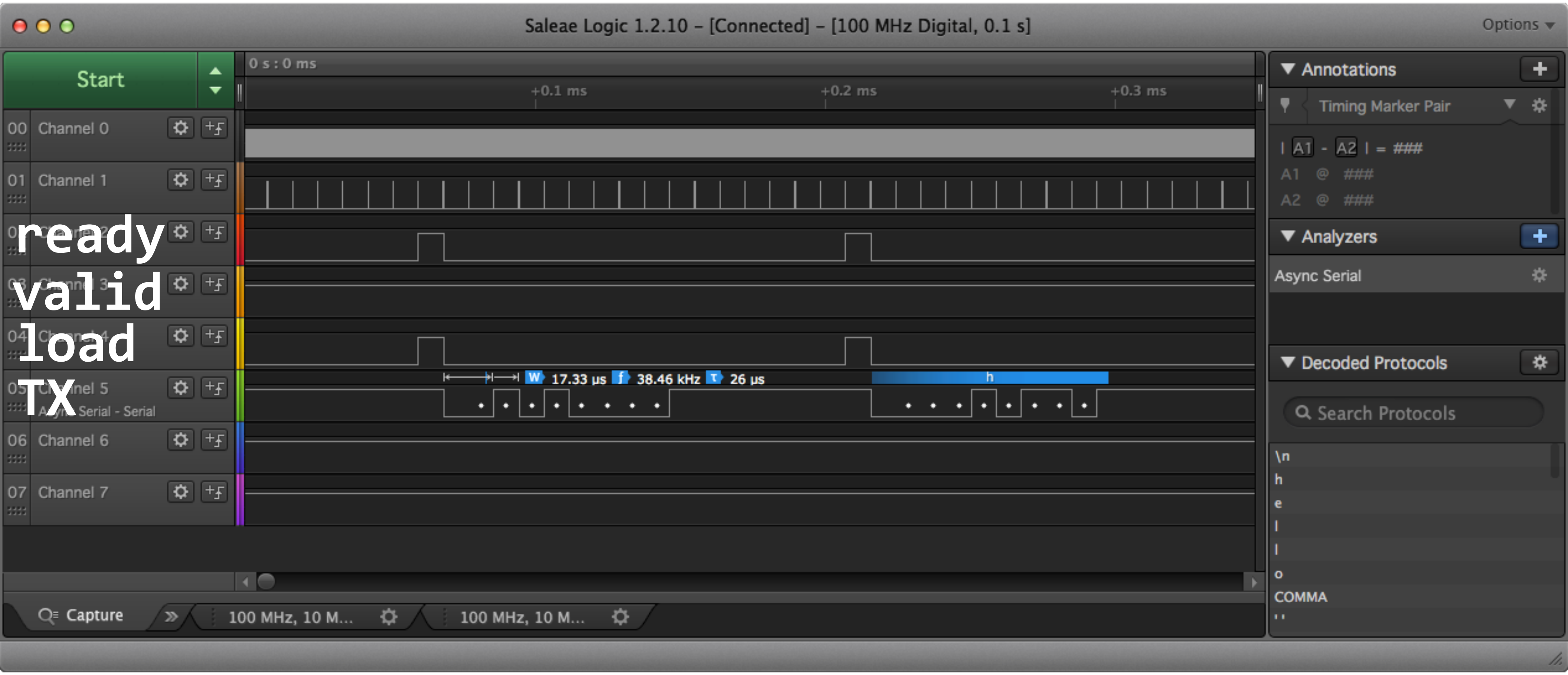
Saleae Logic 1.2.10 – [Connected] – [100 MHz Digital, 0.1 s]

W 8.59 µs f 115.3 kHz T 8.67 µs

fsm.py

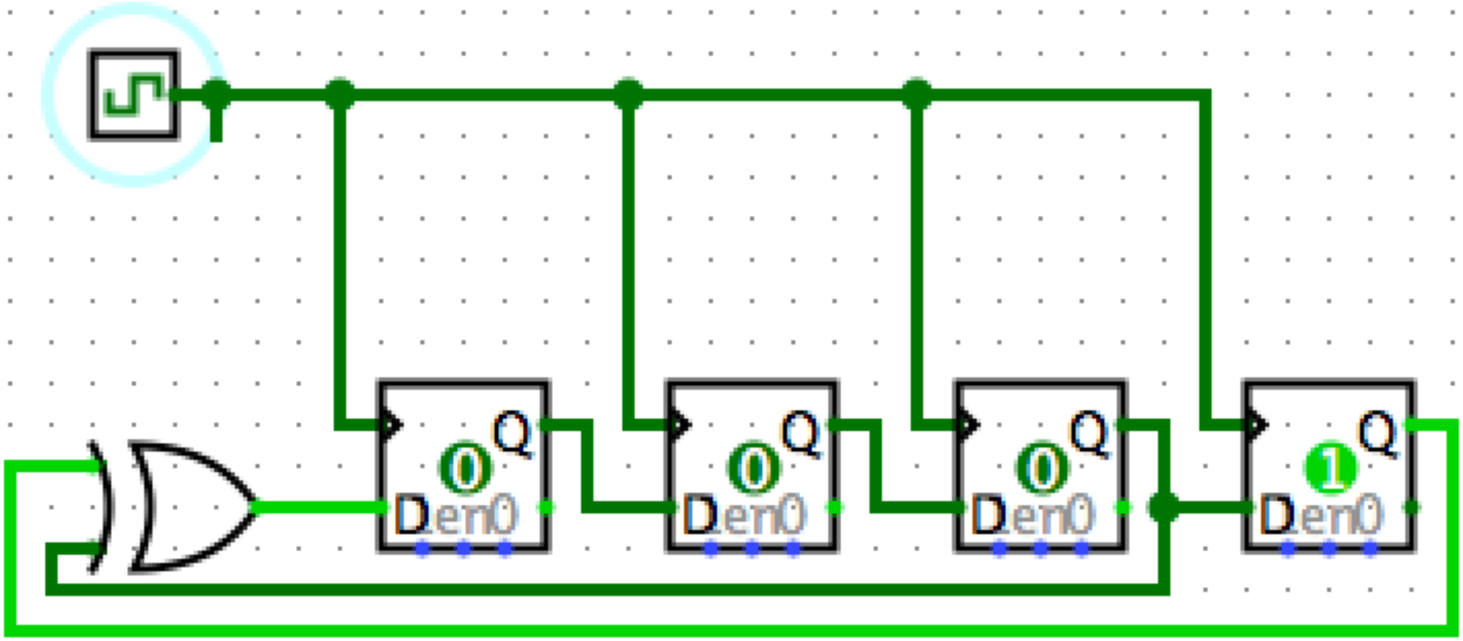

Saleae Logic 1.2.10 – [Connected] – [100 MHz Digital, 0.1 s]
Options
Start
+0.1 ms
+0.2 ms
+0.3 ms
+0.4 ms
00 Channel 0
01 Channel 1
02
03
04 Channel 4
05 Channel 5
06 Channel 6
07 Channel 7
run
done
count
Annotations
Timing Marker Pair
| A1 - A2 | = ###
A1 @ ###
A2 @ ###
Analyzers
Decoded Protocols
Search Protocols
Capture
100 MHz, 10 M...
100 MHz, 10 M...

uart.py


Saleae Logic 1.2.10 – [Connected] – [100 MHz Digital, 0.1 s]
Options
Start
0 s : 0 ms
+0.1 ms
+0.2 ms
+0.3 ms
00 Channel 0
01 Channel 1
ready
valid
load
TX
Async Serial - Serial
06 Channel 6
07 Channel 7
W 17.33 µs
f 38.46 kHz
τ 26 µs
h
Annotations
Timing Marker Pair
| A1 - A2 | = ###
A1 @ ###
A2 @ ###
Analyzers
Async Serial
Decoded Protocols
Search Protocols
\n
h
e
l
l
o
COMMA
' '
Capture
100 MHz, 10 M...
100 MHz, 10 M...

LFSR

Linear Feedback Shift Register



0001
1000
0100
0010
1001
1100
0110
1011
0101
1010
1101
1110
1111
0111
0011
0001

```
BITS,TAPS
3,"3,2"
4,"4,3"
5,"5,3"
6,"6,5"
7,"7,6"
8,"8,6,5,4"
9,"9,5"
10,"10,7"
11,"11,9"
12,"12,6,4,1"
13,"13,4,3,1"
14,"14,5,3,1"
15,"15,14"
16,"16,15,13,4"
17,"17,14"
18,"18,11"
19,"19,6,2,1"
20,"20,17"
21,"21,19"
22,"22,21"
23,"23,18"
24,"24,23,22,17"
25,"25,22"
26,"26,6,2,1"
27,"27,5,2,1"
28,"28,25"
29,"29,27"
30,"30,6,4,1"
31,"31,28"
32,"32,22,2,1"
```