Memories

V1.0

- Usage of Memories
  - Addressable memories
  - FIFO
  - Stream FIFOs
- Workshop

Franck Jullien

@fjullien06

https://github.com/fjullien

Memories – addressable RAM/ROM

```
class Memory(Special):
    def __init__(self, width, depth, init=None, name=None):
```

```
get_port(self,
         write_capable  = False,
         async_read     = False,
         has_re         = False,
         we_granularity = 0,
         mode           = WRITE_FIRST,
         clock_domain   = "sys"
)
```

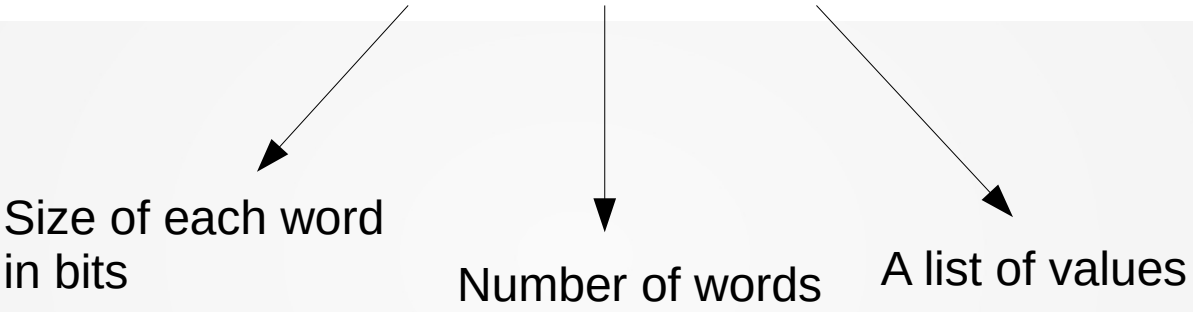
Memories – example

```
init_data = [
    0xA, 0x11223344,
    0x0, 0x66998855,
    0x1, 0x00000000,
    0x4, 0x00000044,
    0x8, 0x00000000,
    0x5, 0x0000000A,
    0x1, 0xFF000000
]

mem = Memory(32, len(init_data), init=init_data)
self.rport = rport = mem.get_port(write_capable=False)
self.wport = wport = mem.get_port(write_capable=True)
self.specials += mem, rport, wport
```

- Interface:
  - rport.**adr**, rport.**dat_r**
  - wport.**adr**, wport.**dat_w**, wport.**dat_r**, wport.**we**

Memories – addressable RAM/ROM

```
class Memory(Special):
    def __init__(self, width, depth, init=None, name=None):
```



Memories – get_port

```
get_port(self,
         write_capable  = False,
         async_read     = False,
         has_re         = False,
         we_granularity = 0,
         mode           = WRITE_FIRST,
         clock_domain   = "sys"
)
```

Port will have a **dat_w** and a **we** signals if **True**

Memories – get_port

```
get_port(self,
         write_capable  = False,
         async_read     = False,
         has_re         = False,
         we_granularity = 0,
         mode           = WRITE_FIRST,
         clock_domain   = "sys"
)
```

Port will have a **dat_w** and a **we** signals if **True**

Read data are immediately available if **async_read** is **True**. If False, reads are synchronous (value on the next clock cycle).

Memories – get_port

```
get_port(self,
         write_capable  = False,
         async_read     = False,
         has_re         = False,
         we_granularity = 0,
         mode           = WRITE_FIRST,
         clock_domain   = "sys"
)
```

Port will have a **dat_w** and a **we** signals if **True**

Read data are immediately available if **async_read** is **True**. If False, reads are synchronous (value on the next clock cycle).

Port will have a **re** signal to allow output data update (only in synchronous mode) if **True**

Memories – get_port

```
get_port(self,
         write_capable  = False,
         async_read     = False,
         has_re         = False,
         we_granularity = 0,
         mode           = WRITE_FIRST,
         clock_domain   = "sys"
)
```

Port will have a **dat_w** and a **we** signals if **True**

Read data are immediately available if **async_read** is **True**. If False, reads are synchronous (value on the next clock cycle).

Port will have a **re** signal to allow output data update (only in synchronous mode) if **True**

Choose how many **dat_w** bits are masked by each **we** bit

Memories – get_port

```
get_port(self,
         write_capable  = False,
         async_read     = False,
         has_re         = False,
         we_granularity = 0,
         mode           = WRITE_FIRST,
         clock_domain   = "sys"
)
```

Port will have a **dat_w** and a **we** signals if **True**

Read data are immediately available if **async_read** is **True**. If False, reads are synchronous (value on the next clock cycle).

Chose which **dat_r** value is given **during a write** (ignored for asynchronous ports)

Port will have a **re** signal to allow output data update (only in synchronous mode) if **True**

Choose how many **dat_w** bits are masked by each **we** bit

Memories – get_port

```
get_port(self,
         write_capable  = False,
         async_read     = False,
         has_re         = False,
         we_granularity = 0,
         mode           = WRITE_FIRST,
         clock_domain   = "sys"
)
```

Port will have a **dat_w** and a **we** signals if **True**

Read data are immediately available if **async_read** is **True**. If False, reads are synchronous (value on the next clock cycle).

Chose which **dat_r** value is given **during a write** (ignored for asynchronous ports)

Ports can have different clock domain

Port will have a **re** signal to allow output data update (only in synchronous mode) if **True**

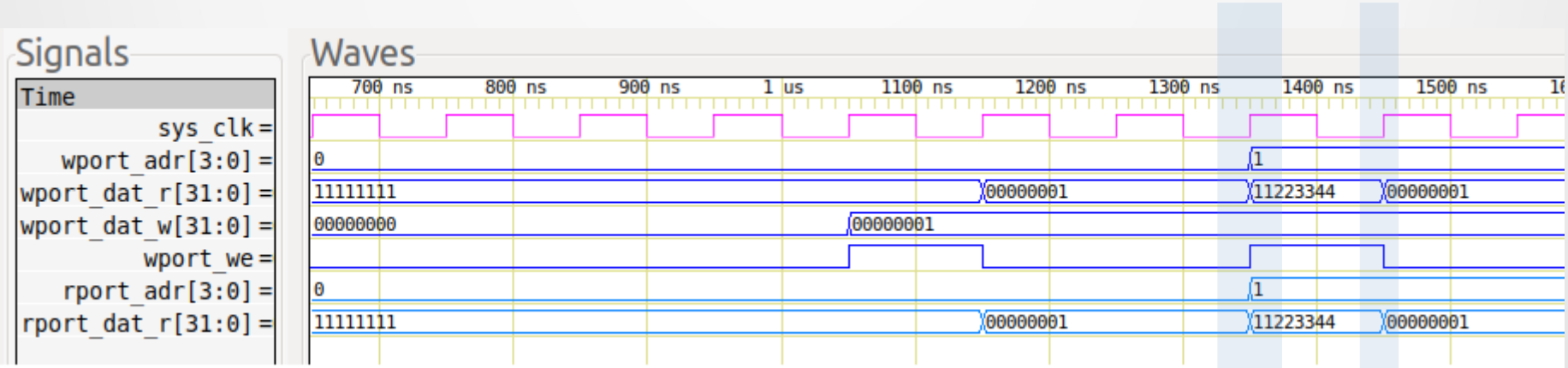
Choose how many **dat_w** bits are masked by each **we** bit

Memories – get_port

```
get_port(self,
         write_capable  = False,
         async_read     = True,
         has_re         = False,
         we_granularity = 0,
         mode           = WRITE_FIRST,
         clock_domain   = "sys"
)
```

Read/Write behavior is always the same (mode is ignored)

dat_w writes are latched



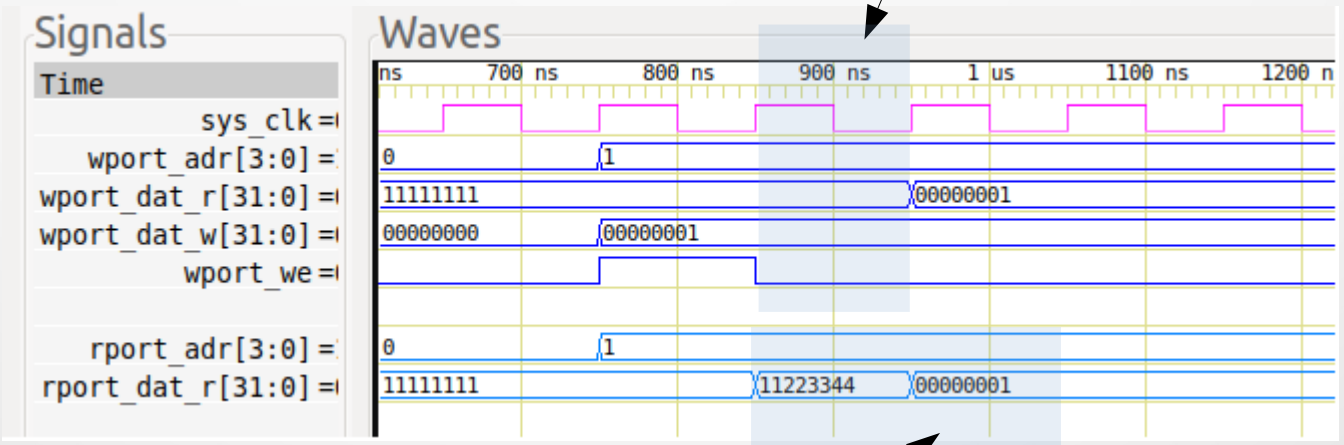
dat_r changes as soon as **adr** changes

Memories – get_port

```
get_port(self,
         write_capable  = True,
         async_read     = False,
         has_re         = False,
         we_granularity = 0,
         mode           = NO_CHANGE,
         clock_domain   = "sys"
)
```

The data read signal keeps its previous value on a write.

After the write cycle, **dat_r** keeps its previous value, then the written value

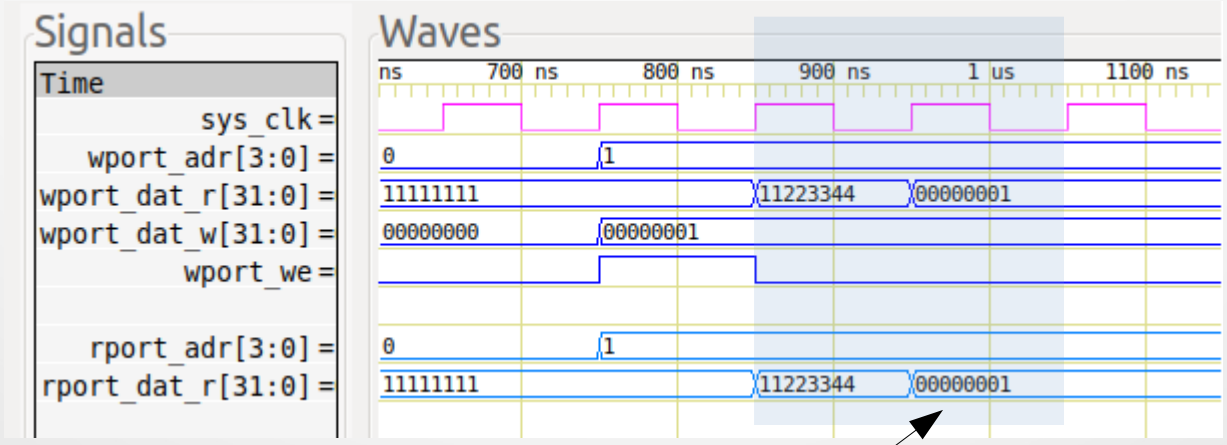


dat_r is value from the old address 1 then the written value

Memories – get_port

```
get_port(self,
         write_capable  = True,
         async_read     = False,
         has_re         = False,
         we_granularity = 0,
         mode           = READ_FIRST,
         clock_domain   = "sys"
)
```

During a write, the previous value is read

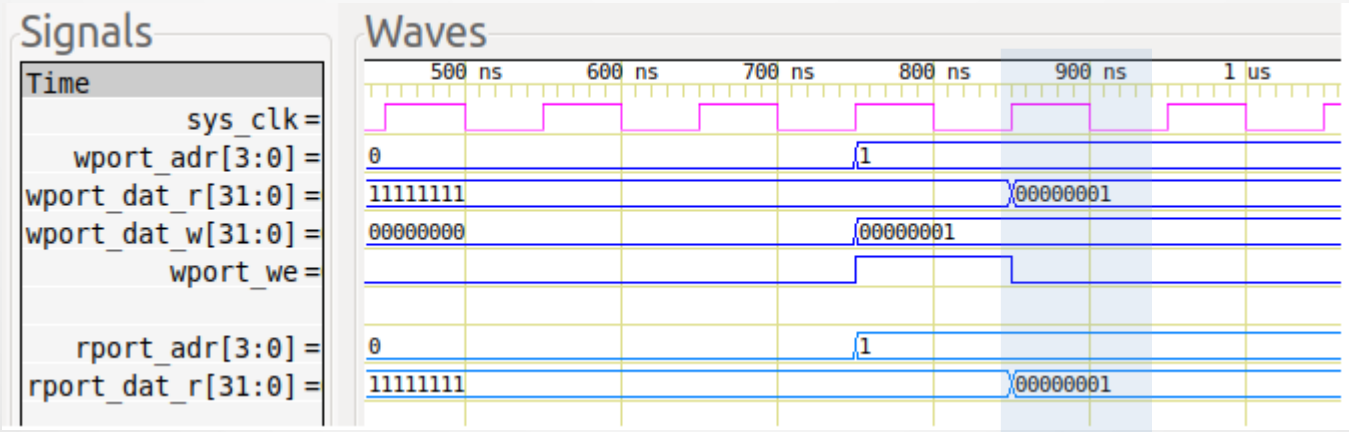


dat_r is value from the old address 1 then the written value

Memories – get_port

```
get_port(self,
         write_capable  = True,
         async_read     = False,
         has_re         = False,
         we_granularity = 0,
         mode           = WRITE_FIRST,
         clock_domain   = "sys"
)
```

The written value is returned



dat_r is value from the the written value

Memories – FIFOs

- SyncFIFO → Same clock on both sides
- SyncFIFOBuffered → Same as SyncFIFO but uses synchronous output
- AsyncFIFO → Different clocks on both sides
- AsyncFIFOBuffered → Same as AsyncFIFO but uses synchronous output

- They are Modules and must be added as submodules

Memories – FIFOs

- Parameters:
  - width → data width in bits
  - depth → number of data words
  - fwft → First Word Fall Through (only for SyncFIFO)
- Interfaces:
  - din → data input
  - we → write enable
  - writable → ‘1’ if the fifo is not full
  - dout → data output
  - re → read enable
  - readable → ‘1’ if the fifo is not empty
  - level → number of unread entries
  - replace → replaces the last entry written into the FIFO

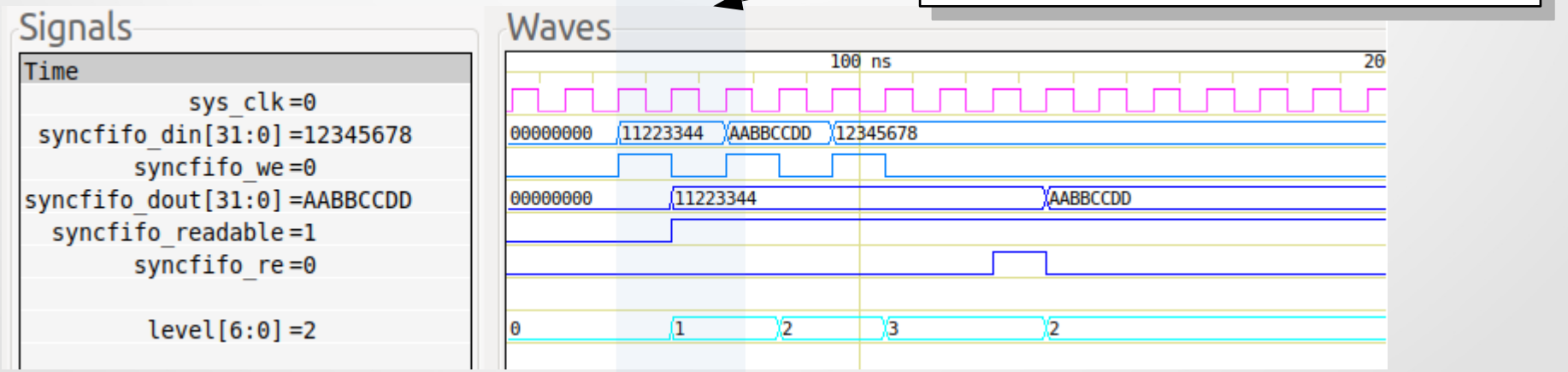
Memories – SyncFIFO

```
def test(dut):
    for i in range(5):
        yield
    yield from dut.write(0x11223344)
    yield from dut.write(0xaabbccdd)
    yield from dut.write(0x12345678)
    yield
    yield from dut.read()
    for i in range(30):
        yield

def main():
    dut = SyncFIFO(width=32, depth=64, fwft=True)
    run_simulation(dut, test(dut), vcd_name="sim.vcd")
```

FIFO classes have read and write methods for simulation

```
def read(self):
    """Read method for simulation."""
    value = (yield self.dout)
    yield self.re.eq(1)
    yield
    yield self.re.eq(0)
    yield
    return value
```

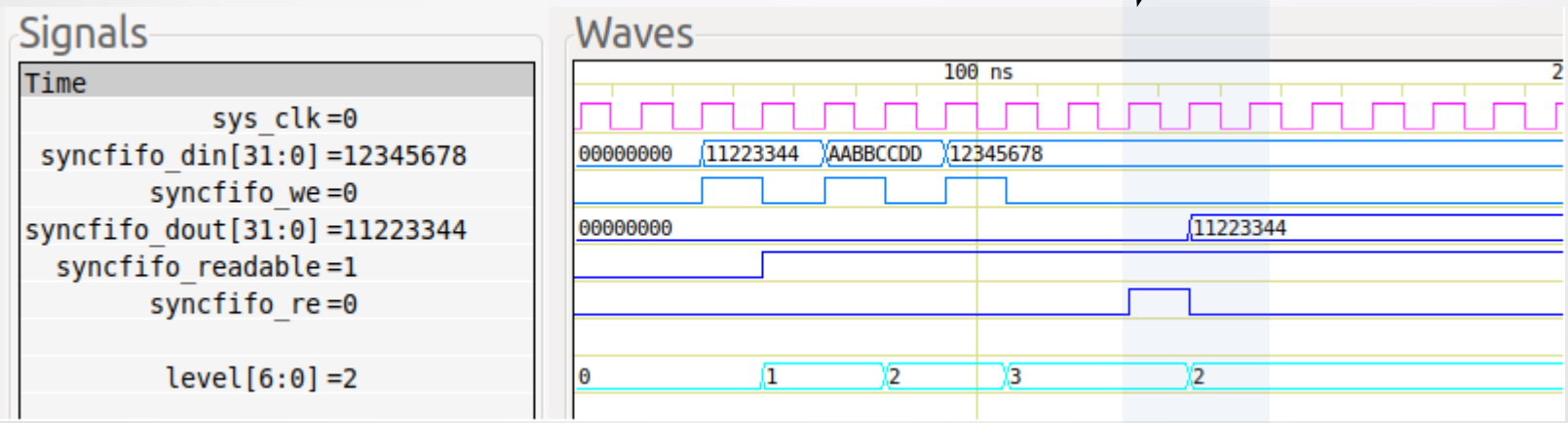
The data is available immediately



Memories – SyncFIFO

```
def test(dut):
    for i in range(5):
        yield
    yield from dut.write(0x11223344)
    yield from dut.write(0xaabbccdd)
    yield from dut.write(0x12345678)
    yield
    yield from dut.read()
    for i in range(30):
        yield

def main():
    dut = SyncFIFO(width=32, depth=64, fwft=False)
    run_simulation(dut, test(dut), vcd_name="sim.vcd")
```

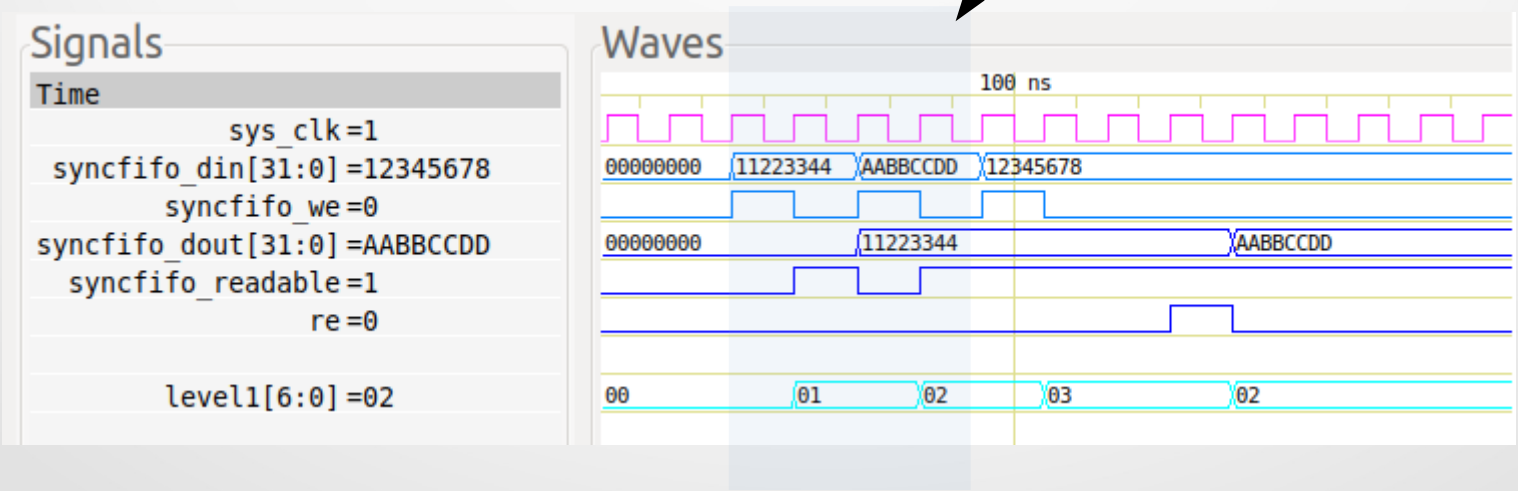
The data is available when requested



Memories – SyncFIFOBuffered

```python
def test(dut):
    for i in range(5):
        yield
    yield from dut.write(0x11223344)
    yield from dut.write(0xaabbccdd)
    yield from dut.write(0x12345678)
    yield
    yield from dut.read()
    for i in range(30):
        yield

def main():
    dut = SyncFIFOBuffered(width=32, depth=64)
    run_simulation(dut, test(dut), vcd_name="sim.vcd")
```

Same as SyncFIFO with fwft=True but with an extra latency cycle

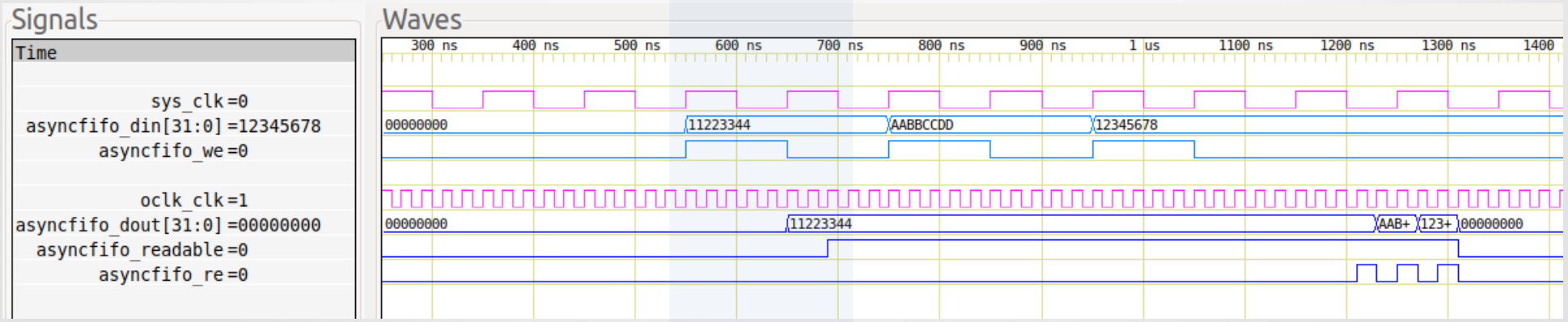


Memories – AsyncFIFO

```python
def test_write(dut):
    for i in range(5):
        yield
    yield from dut.write(0x11223344)
    yield from dut.write(0xaabbccdd)
    yield from dut.write(0x12345678)
    for i in range(30):
        yield

def test_read(dut):
    for i in range(60):
        yield
    yield from dut.read()
    yield from dut.read()
    yield from dut.read()
    for i in range(30):
        yield

def main():
    dut = AsyncFIFO(width=32, depth=64)
    dut = ClockDomainsRenamer({"write": "sys", "read" : "oclk"})(dut)

    generators = {
        "sys"  : test_write(dut),
        "oclk" : test_read(dut)
    }

    run_simulation(dut, generators, clocks={"sys": 1e9/10e6, "oclk" : 1e9/50e6}, vcd_name="sim.vcd")
```

Clock domains have to be renamed

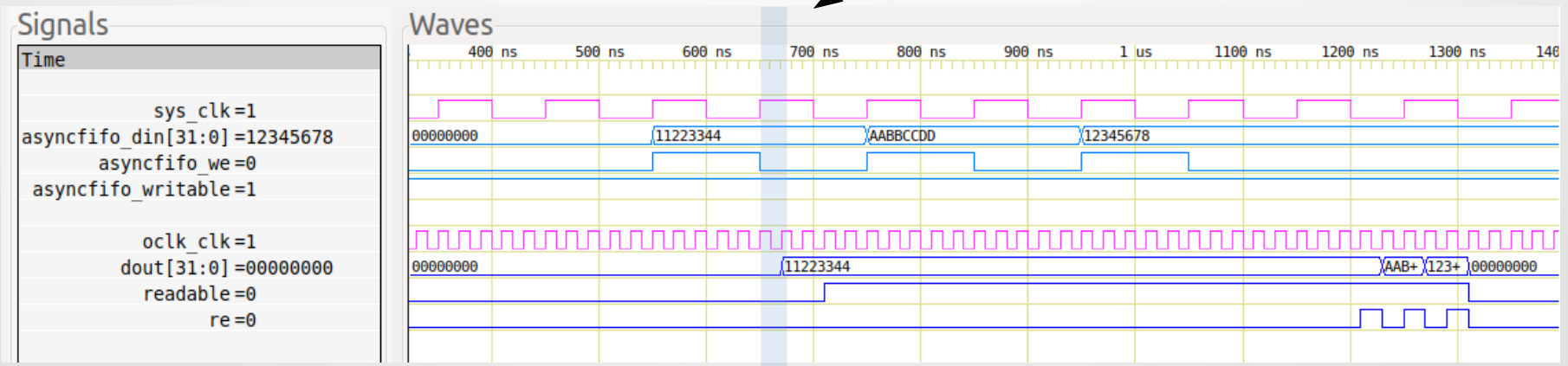
Same as SyncFIFO with fwft=True



Memories – AsyncFIFOBuffered

```python
def test_write(dut):
    for i in range(5):
        yield
    yield from dut.write(0x11223344)
    yield from dut.write(0xaabbccdd)
    yield from dut.write(0x12345678)
    for i in range(30):
        yield

def test_read(dut):
    for i in range(60):
        yield
    yield from dut.read()
    yield from dut.read()
    yield from dut.read()
    for i in range(30):
        yield

def main():
    dut = AsyncFIFOBuffered(width=32, depth=64)
    dut = ClockDomainsRenamer({"write": "sys", "read" : "oclk"})(dut)

    generators = {
        "sys"  : test_write(dut),
        "oclk" : test_read(dut)
    }

    run_simulation(dut, generators, clocks={"sys": 1e9/10e6, "oclk" : 1e9/50e6}, vcd_name="sim.vcd")
```

Same as AsyncFIFO with one extra cycle



Memories – Stream FIFO

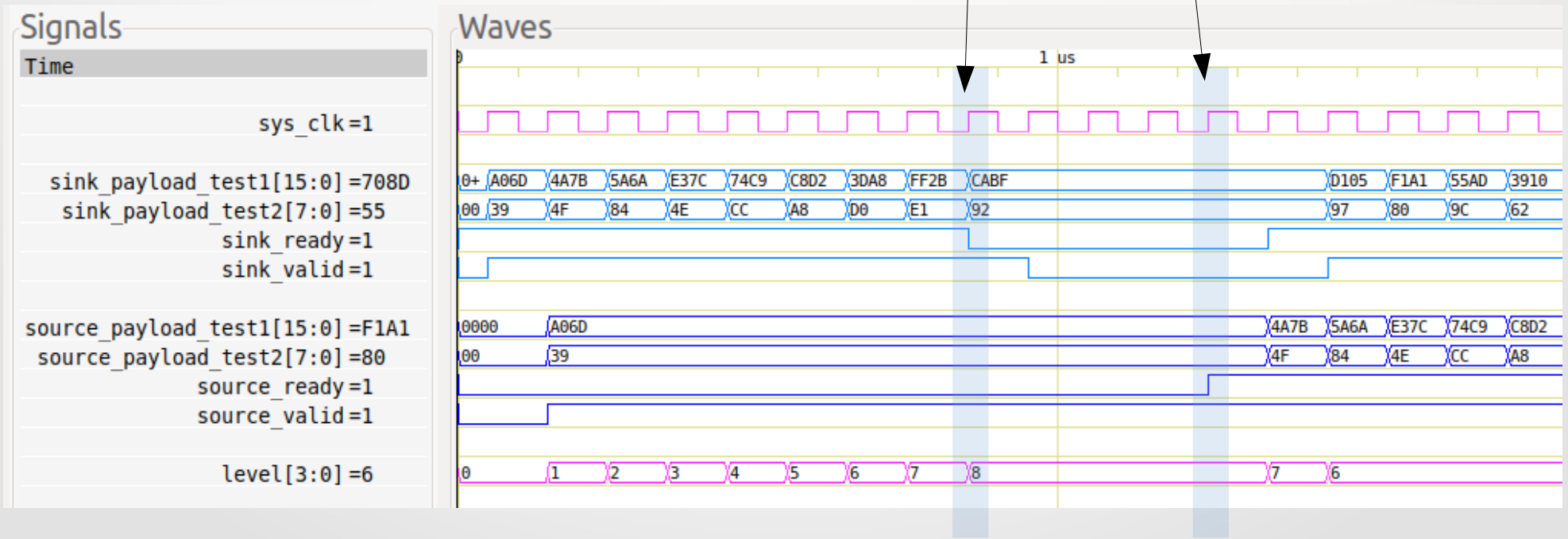
- stream.SyncFIFO, stream.AsyncFIFO
- Sync/AsyncFIFO**Buffered** variant replaced by a parameter
- Data width is replaced by layout
- Ready/Valid signals replace readable/writable
  - Sink → Ready while there is free space
  - Source → Valid when the FIFO not empty
- They are Modules and must be added as submodules

Memories – SyncFIFO

```python
from litex.soc.interconnect import stream

layout_test = [
    ("test1", 16),
    ("test2", 8)
]

def test_write(dut):
    #...

def test_read(dut):
    #...

def main():
    dut = stream.SyncFIFO(layout_test, depth=8)

    generators = {
        "sys"  : [test_write(dut), test_read(dut)]
    }

    run_simulation(dut, generators, clocks={"sys": 1e9/10e6}, vcd_name="sim.vcd")
```

The FIFO is full.
sink.ready goes low

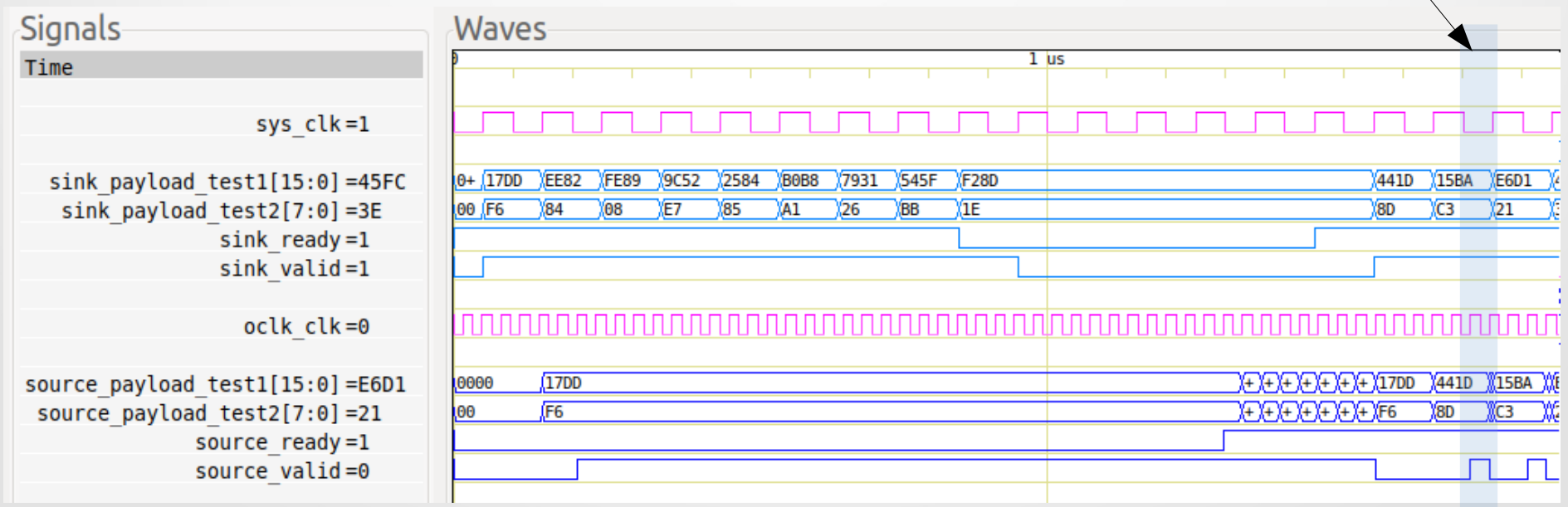
When the source gets the ready signal data can be extracted From the FIFO



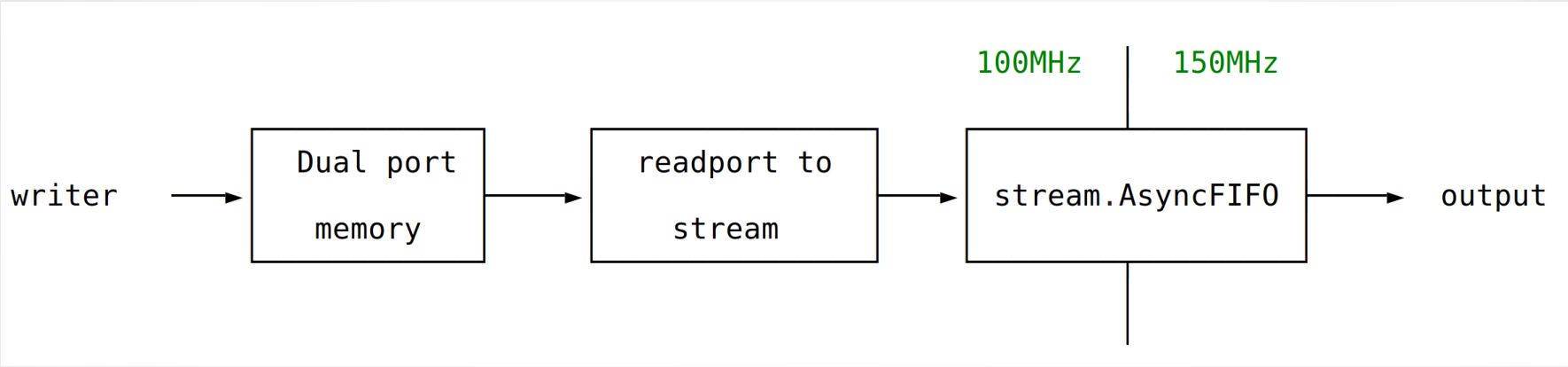
Memories – AsyncFIFO

```python
from litex.soc.interconnect import stream

layout_test = [
    ("test1", 16),
    ("test2", 8)
]


def test_write(dut):
    #...

def test_read(dut):
    #...

def main():
    dut = stream.AsyncFIFO(layout_test, depth=8)
    dut = ClockDomainsRenamer({"write" : "sys", "read" : "oclk"})(dut)
    generators = {
        "sys"  : test_write(dut),
        "oclk" : test_read(dut)
    }

    run_simulation(dut, generators, clocks={"sys": 1e9/10e6, "oclk": 1e9/30e6}, vcd_name="sim.vcd")
```

Always use ready & valid as an acknowledge



Workshop – extra 1



- Create a stream clocked at 150MHz
- Data from an initialized memory
- Data can be updated from a writer port