FPGA Development for Radar, Radio-Astronomy and Communications

THE RADAR MASTERS COURSE

Dept. Electrical Engineering, University of Cape Town
Private Bag, Rondebosch, 7701, South Africa

http://www.rrsg.uct.ac.za

Presented by John-Philip Taylor

Convened by Dr Stephen Paine

Day 4 – 2 May 2022

Outline

Advanced Clocking

Clock Domains

Mutual Exclusion and Arbitration

Caching Systems

Flow Control

Practical – NCO and DDC

Numerically-controlled Oscillator

Outline

Advanced Clocking

Clock Domains

Mutual Exclusion and Arbitration

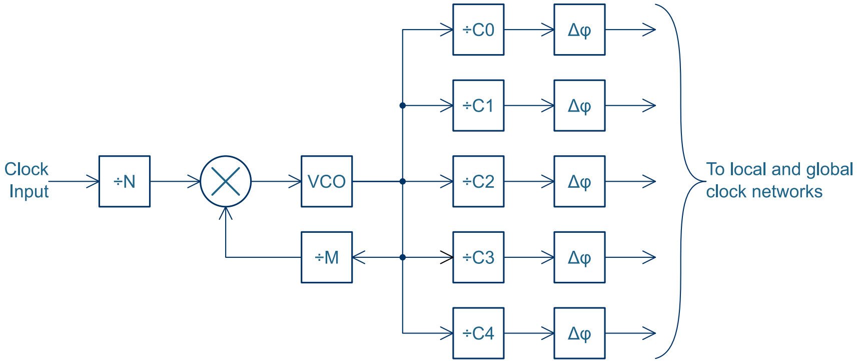
Caching Systems

Flow Control

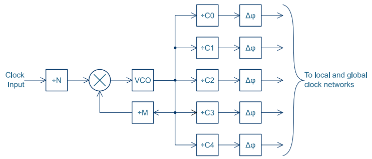
Practical – NCO and DDC

Numerically-controlled Oscillator

Phase-locked Loop

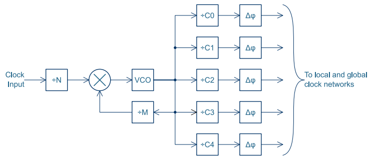


Phase-locked Loop



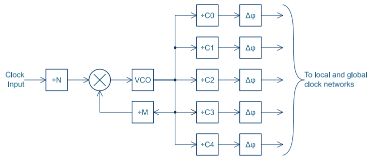
- Useful for generating a range of related clocks
- Output clock rising and falling edges can be set, independently, to a resolution equal to the VCO period, which is typically about 1 ns $\pm$ 500 ps
- All output clocks can be considered to be in the same clock domain
- Noise sensitive applications should avoid PLLs due to high phase noise

Phase-locked Loop



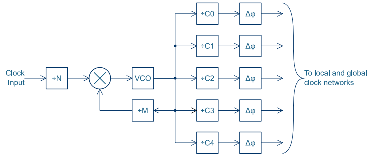
- Useful for generating a range of related clocks
- Output clock rising and falling edges can be set, independently, to a resolution equal to the VCO period, which is typically about 1 ns $\pm$ 500 ps
- All output clocks can be considered to be in the same clock domain
- Noise sensitive applications should avoid PLLs due to high phase noise

Phase-locked Loop



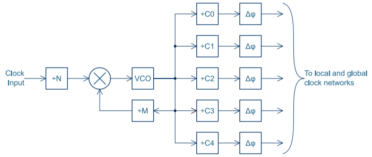
- Useful for generating a range of related clocks
- Output clock rising and falling edges can be set, independently, to a resolution equal to the VCO period, which is typically about 1 ns $\pm$ 500 ps
- All output clocks can be considered to be in the same clock domain
- Noise sensitive applications should avoid PLLs due to high phase noise

Phase-locked Loop



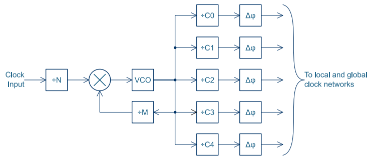
- Useful for generating a range of related clocks
- Output clock rising and falling edges can be set, independently, to a resolution equal to the VCO period, which is typically about 1 ns $\pm$ 500 ps
- All output clocks can be considered to be in the same clock domain
- Noise sensitive applications should avoid PLLs due to high phase noise

PLL Compensation



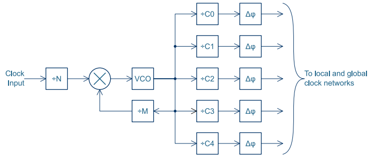
- **Direct**: No compensation, which results in better jitter performance
- **Normal**: The clock at the register is phase-aligned with the input clock pin
- **Source Synchronous**: The PLL ensures that the data delay from pin to register and the apparent clock delay from pin to register is the same
- **Zero-Delay Buffer**: The output clock pin is phase-aligned with the input clock pin

PLL Compensation



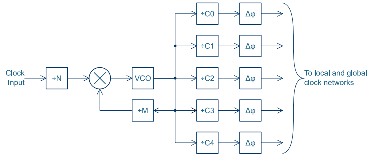
- **Direct**: No compensation, which results in better jitter performance
- **Normal**: The clock at the register is phase-aligned with the input clock pin
- **Source Synchronous**: The PLL ensures that the data delay from pin to register and the apparent clock delay from pin to register is the same
- **Zero-Delay Buffer**: The output clock pin is phase-aligned with the input clock pin

PLL Compensation



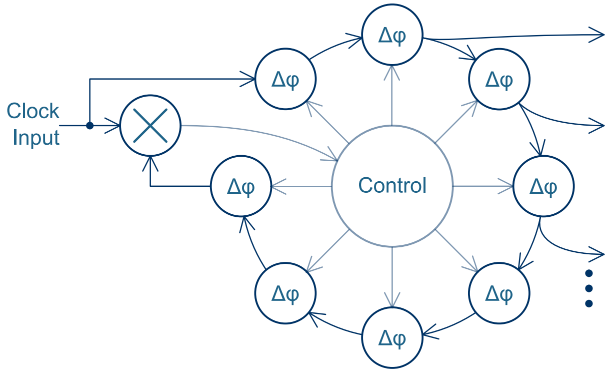
- **Direct**: No compensation, which results in better jitter performance
- **Normal**: The clock at the register is phase-aligned with the input clock pin
- **Source Synchronous**: The PLL ensures that the data delay from pin to register and the apparent clock delay from pin to register is the same
- **Zero-Delay Buffer**: The output clock pin is phase-aligned with the input clock pin

PLL Compensation

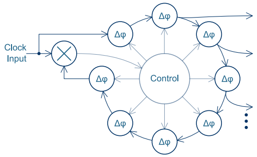


- **Direct**: No compensation, which results in better jitter performance
- **Normal**: The clock at the register is phase-aligned with the input clock pin
- **Source Synchronous**: The PLL ensures that the data delay from pin to register and the apparent clock delay from pin to register is the same
- **Zero-Delay Buffer**: The output clock pin is phase-aligned with the input clock pin

Delay-locked Loop

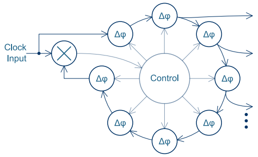


Delay-locked Loop



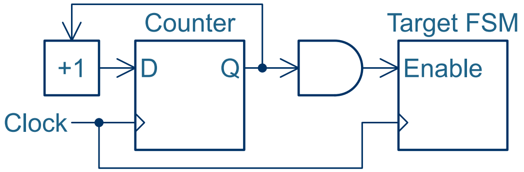
- Useful for performing phase compensation when interfacing to fast peripherals
- All output clocks can be considered to be in the same clock domain as the input clock

Delay-locked Loop

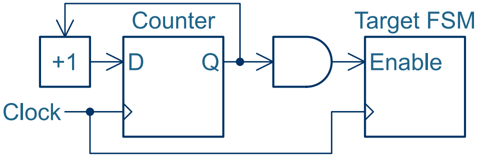


- Useful for performing phase compensation when interfacing to fast peripherals
- All output clocks can be considered to be in the same clock domain as the input clock

Clock-Enable Generated

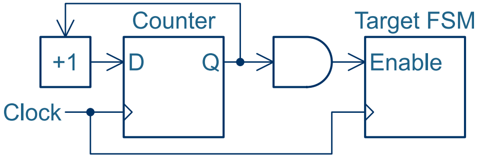


Clock-Enable Generated



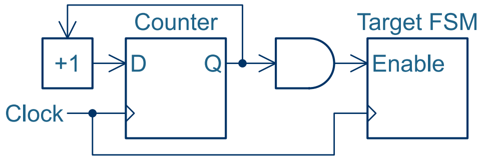
- Useful when mixing fast and slow systems
- Potential to reduce overall device power usage
- Cannot be used for clocking external devices
- The multi-cycle timing requirements must be manually specified

Clock-Enable Generated



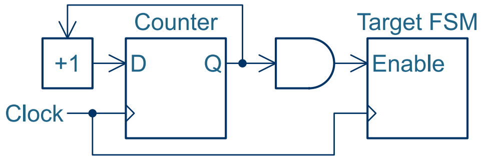
- Useful when mixing fast and slow systems
- Potential to reduce overall device power usage
- Cannot be used for clocking external devices
- The multi-cycle timing requirements must be manually specified

Clock-Enable Generated



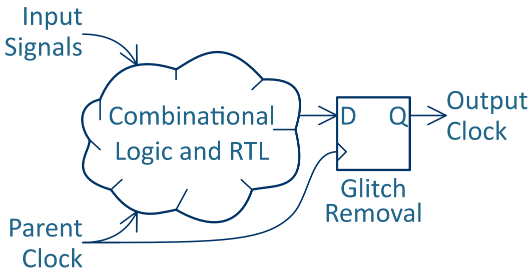
- Useful when mixing fast and slow systems
- Potential to reduce overall device power usage
- Cannot be used for clocking external devices
- The multi-cycle timing requirements must be manually specified

Clock-Enable Generated

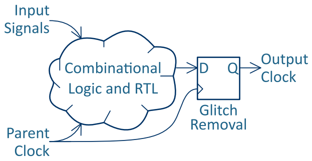


- Useful when mixing fast and slow systems
- Potential to reduce overall device power usage
- Cannot be used for clocking external devices
- The multi-cycle timing requirements must be manually specified

Ripple and Gated Clocks

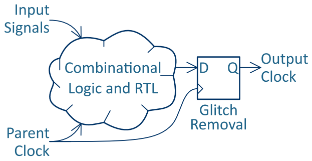


Ripple and Gated Clocks



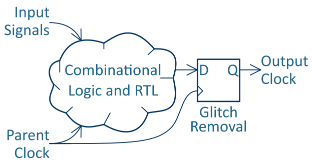
- Unrelated to all other clocks in the system, including the parent clock
- Often necessitates complex clock-domain crossing schemes to the general system clock
- Useful when:
  - Requiring very low power usage
  - The frequency must change often
  - The clock must drive an external device
  - etc.

Ripple and Gated Clocks



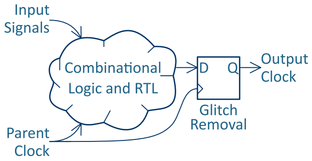
- Unrelated to all other clocks in the system, including the parent clock
- Often necessitates complex clock-domain crossing schemes to the general system clock
- Useful when:
  - Requiring very low power usage
  - The frequency must change often
  - The clock must drive an external device
  - etc.

Ripple and Gated Clocks



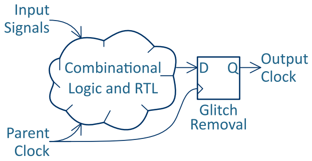
- Unrelated to all other clocks in the system, including the parent clock
- Often necessitates complex clock-domain crossing schemes to the general system clock
- Useful when:
  - Requiring very low power usage
  - The frequency must change often
  - The clock must drive an external device
  - etc.

Ripple and Gated Clocks



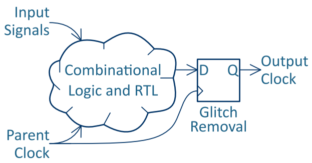
- Unrelated to all other clocks in the system, including the parent clock
- Often necessitates complex clock-domain crossing schemes to the general system clock
- Useful when:
  - Requiring very low power usage
  - The frequency must change often
  - The clock must drive an external device
  - etc.

Ripple and Gated Clocks



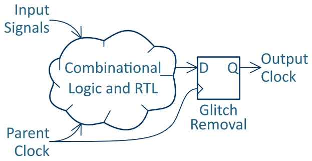
- Unrelated to all other clocks in the system, including the parent clock
- Often necessitates complex clock-domain crossing schemes to the general system clock
- Useful when:
  - Requiring very low power usage
  - The frequency must change often
  - The clock must drive an external device
  - etc.

Ripple and Gated Clocks



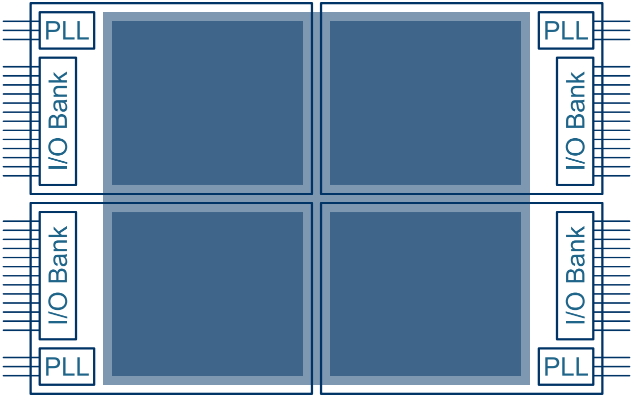
- Unrelated to all other clocks in the system, including the parent clock
- Often necessitates complex clock-domain crossing schemes to the general system clock
- Useful when:
  - Requiring very low power usage
  - The frequency must change often
  - The clock must drive an external device
  - etc.

Ripple and Gated Clocks

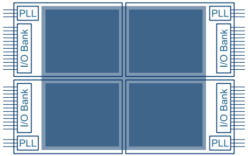


- Unrelated to all other clocks in the system, including the parent clock
- Often necessitates complex clock-domain crossing schemes to the general system clock
- Useful when:
  - Requiring very low power usage
  - The frequency must change often
  - The clock must drive an external device
  - etc.

Clock Regions

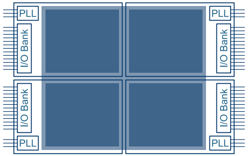


Clock Regions



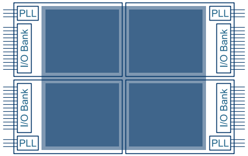
- FPGAs are generally organised into regions, each with its own PLL and I/O bank
- The local clock network is more efficient (and faster) than the global network
- When laying out the PCB, keep fast I/O in the same region as their clock

Clock Regions



- FPGAs are generally organised into regions, each with its own PLL and I/O bank
- The local clock network is more efficient (and faster) than the global network
- When laying out the PCB, keep fast I/O in the same region as their clock

Clock Regions



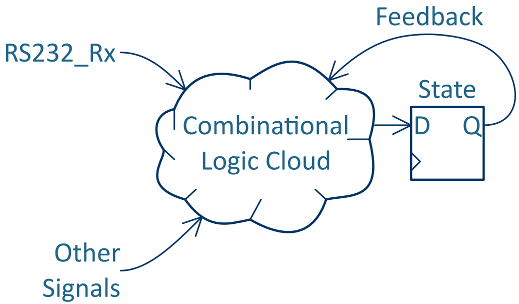
- FPGAs are generally organised into regions, each with its own PLL and I/O bank
- The local clock network is more efficient (and faster) than the global network
- When laying out the PCB, keep fast I/O in the same region as their clock

Outline

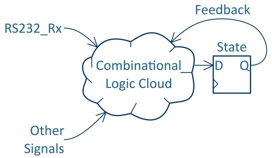
Common Mistake

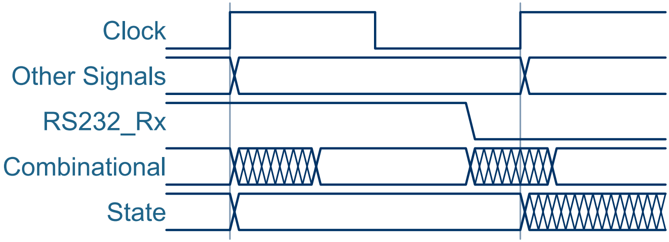
```
module RS232(
  input   ipClk, ipReset,
  input   ipRS232_Rx, // Asynchronous external signal
  output  [7:0]opData
);

... // Definitions, state machine boilerplate, etc.

  Idle: begin
    if(!ipRS232_Rx) begin
      State <= Receiving;
    end
  end

  Receiving: begin
...
```

Common Mistake

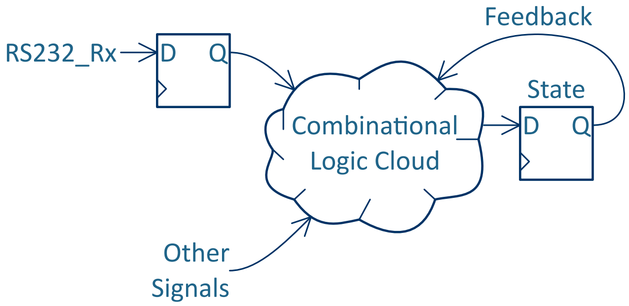


Common Mistake

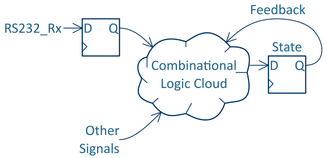


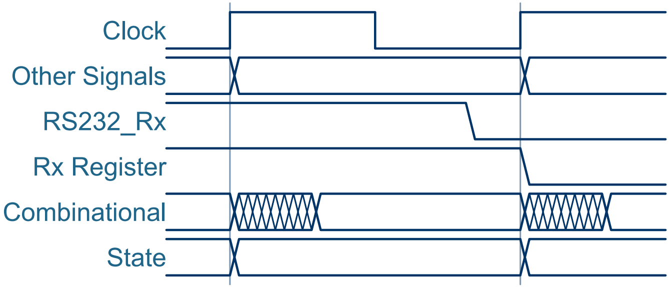


The Solution

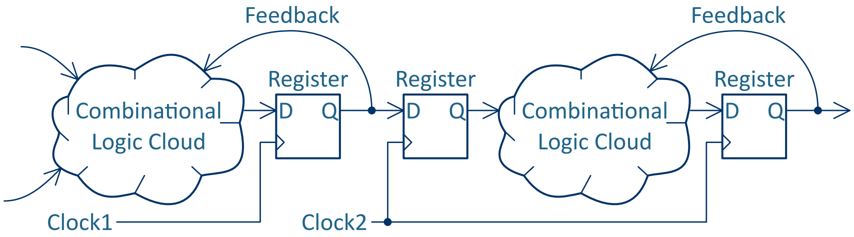


Common Mistake

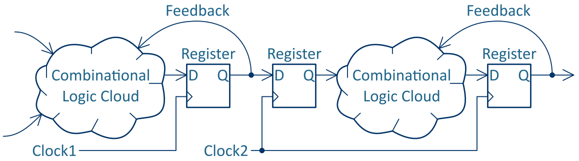




Register Chain

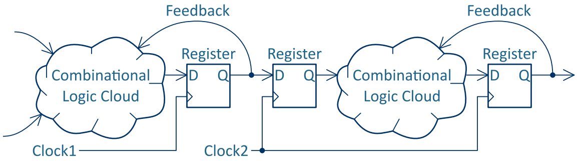


Register Chain



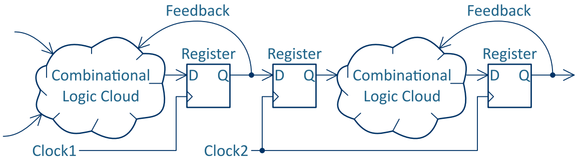
- Essential for crossing asynchronous external signals
- Only useful for crossing one bit at a time (unless the system is tolerant to temporary errors)
- Often used to cross hand-shaking signals
- Works in both directions (fast to slow / slow to fast)

Register Chain



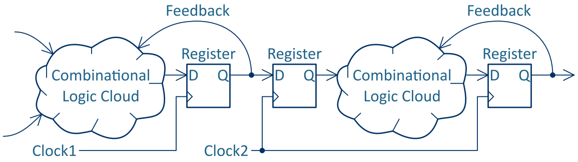
- Essential for crossing asynchronous external signals
- Only useful for crossing one bit at a time (unless the system is tolerant to temporary errors)
- Often used to cross hand-shaking signals
- Works in both directions (fast to slow / slow to fast)

Register Chain



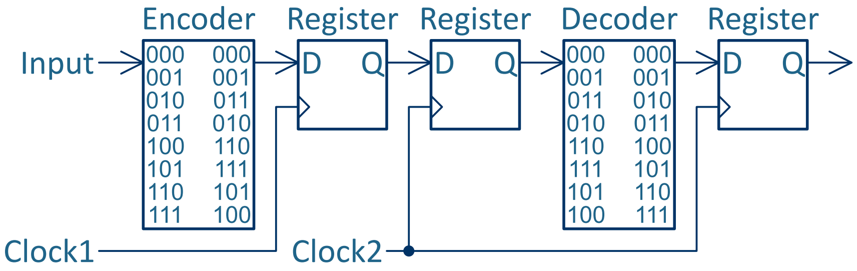
- Essential for crossing asynchronous external signals
- Only useful for crossing one bit at a time (unless the system is tolerant to temporary errors)
- Often used to cross hand-shaking signals
- Works in both directions (fast to slow / slow to fast)

Register Chain

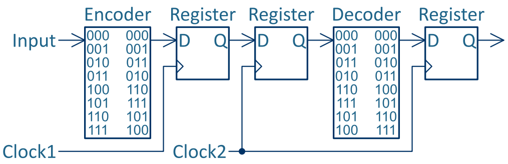


- Essential for crossing asynchronous external signals
- Only useful for crossing one bit at a time (unless the system is tolerant to temporary errors)
- Often used to cross hand-shaking signals
- Works in both directions (fast to slow / slow to fast)

Gray Coding

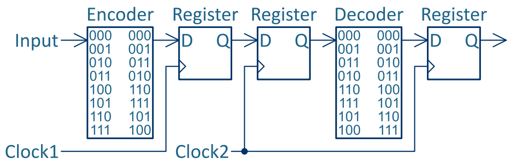

Encoder
Register
Register
Decoder
Register
Input
000 000
001 001
010 011
011 010
100 110
101 111
110 101
111 100
D Q
D Q
000 000
001 001
011 010
010 011
110 100
111 101
101 110
100 111
D Q
Clock1
Clock2

Gray Coding



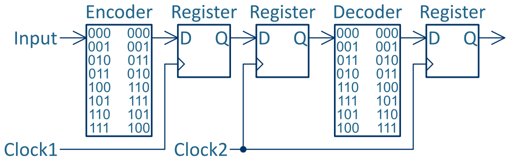
- Encoded such that only one bit changes at a time
- Only works for counter data
  (FIFO queue pointers, rotary encoders, etc.)
- Works in both directions (fast to slow / slow to fast)

Gray Coding



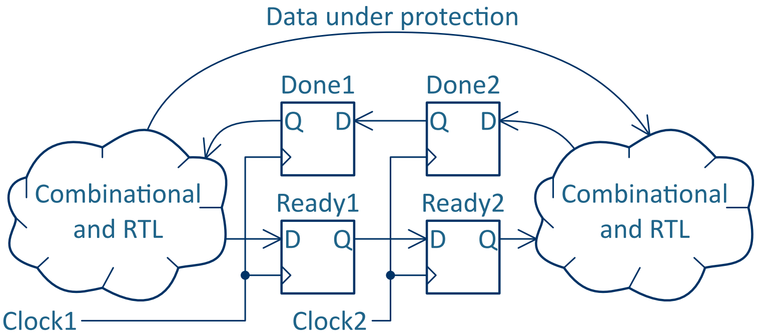
- Encoded such that only one bit changes at a time
- Only works for counter data
  (FIFO queue pointers, rotary encoders, etc.)
- Works in both directions (fast to slow / slow to fast)

Gray Coding

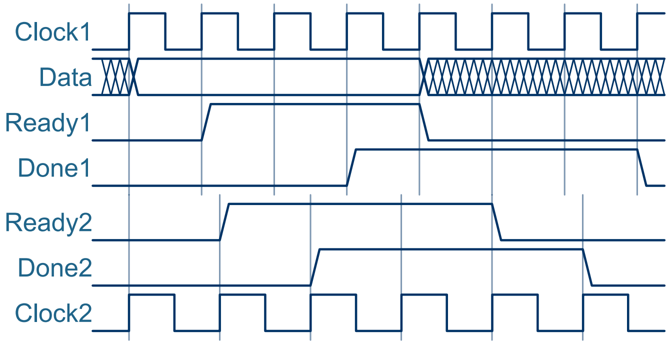


- Encoded such that only one bit changes at a time
- Only works for counter data
  (FIFO queue pointers, rotary encoders, etc.)
- Works in both directions (fast to slow / slow to fast)

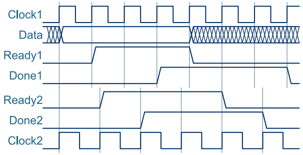
Hand-Shaking



Hand-Shaking

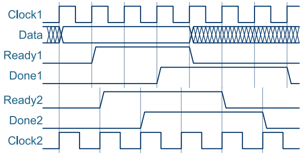


Hand-Shaking



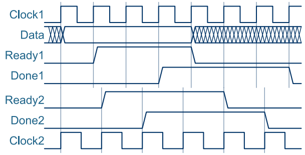
- Source must guarantee data stability while the handshaking is taking place
- Not time-efficient: need to wait for the entire transaction sequence to finish before starting the next one
- Use only for data that does not need to be crossed often
- Works in both directions (fast to slow / slow to fast)

Hand-Shaking



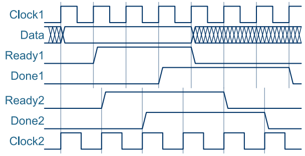
- Source must guarantee data stability while the handshaking is taking place
- Not time-efficient: need to wait for the entire transaction sequence to finish before starting the next one
- Use only for data that does not need to be crossed often
- Works in both directions (fast to slow / slow to fast)

Hand-Shaking



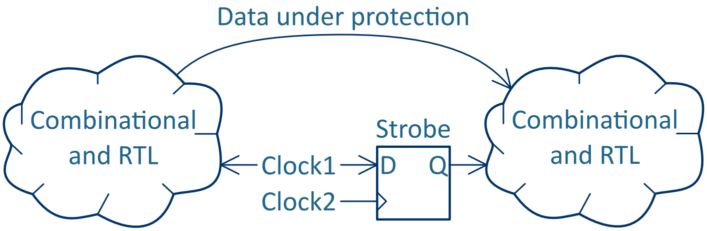
- Source must guarantee data stability while the handshaking is taking place
- Not time-efficient: need to wait for the entire transaction sequence to finish before starting the next one
- Use only for data that does not need to be crossed often
- Works in both directions (fast to slow / slow to fast)

Hand-Shaking

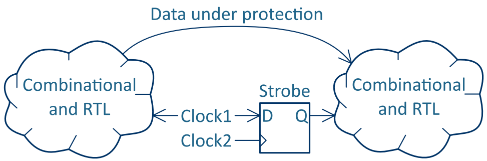


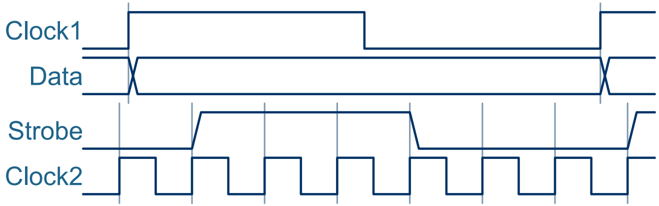
- Source must guarantee data stability while the handshaking is taking place
- Not time-efficient: need to wait for the entire transaction sequence to finish before starting the next one
- Use only for data that does not need to be crossed often
- Works in both directions (fast to slow / slow to fast)

Strobe-based

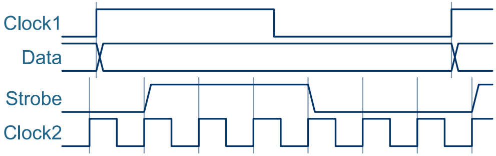


Strobe-based



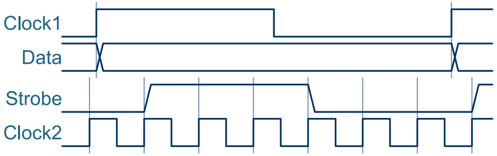


Strobe-based



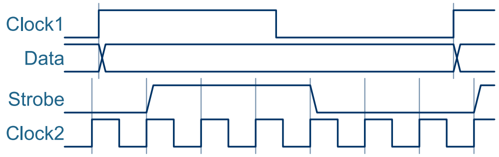
- The data is valid for as long as the strobe is high
- As fast as the source clock
- The strobe edges can be used to perform flow-control
- Only works when the destination clock is much faster than the source clock
- Note that the source “clock” can be a strobe signal generated by the source state machine...

Strobe-based



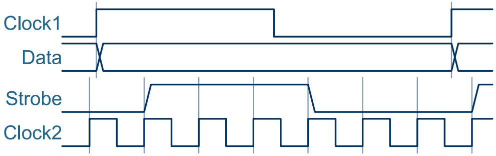
- The data is valid for as long as the strobe is high
- As fast as the source clock
- The strobe edges can be used to perform flow-control
- Only works when the destination clock is much faster than the source clock
- Note that the source “clock” can be a strobe signal generated by the source state machine...

Strobe-based



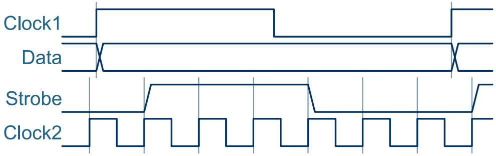
- The data is valid for as long as the strobe is high
- As fast as the source clock
- The strobe edges can be used to perform flow-control
- Only works when the destination clock is much faster than the source clock
- Note that the source "clock" can be a strobe signal generated by the source state machine...

Strobe-based



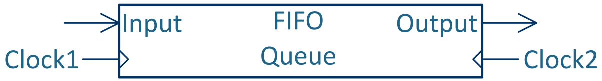
- The data is valid for as long as the strobe is high
- As fast as the source clock
- The strobe edges can be used to perform flow-control
- Only works when the destination clock is much faster than the source clock
- Note that the source “clock” can be a strobe signal generated by the source state machine...

Strobe-based



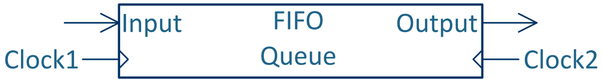
- The data is valid for as long as the strobe is high
- As fast as the source clock
- The strobe edges can be used to perform flow-control
- Only works when the destination clock is much faster than the source clock
- Note that the source “clock” can be a strobe signal generated by the source state machine...

FIFO Queues



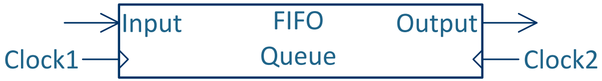
- Flow-control signals are crossed separately
- Control signal crossing latency can be hidden in the length of the queue
- Mixed-width RAM can be used to keep the data-rate constant across different frequencies (especially useful in DDR-based external memory)

FIFO Queues



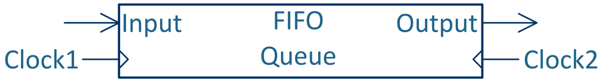
- Flow-control signals are crossed separately
- Control signal crossing latency can be hidden in the length of the queue
- Mixed-width RAM can be used to keep the data-rate constant across different frequencies (especially useful in DDR-based external memory)

FIFO Queues



- Flow-control signals are crossed separately
- Control signal crossing latency can be hidden in the length of the queue
- Mixed-width RAM can be used to keep the data-rate constant across different frequencies (especially useful in DDR-based external memory)

FIFO Queues



- Flow-control signals are crossed separately
- Control signal crossing latency can be hidden in the length of the queue
- Mixed-width RAM can be used to keep the data-rate constant across different frequencies (especially useful in DDR-based external memory)

Clock Groups

- Unrelated clock domains must be kept separate in the timing constraints as well
- Remember clock groups from Day 2?

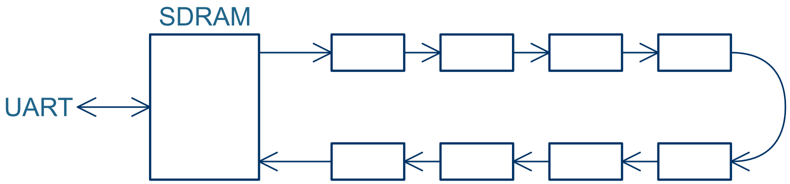
Clock Groups

- Unrelated clock domains must be kept separate in the timing constraints as well
- Remember clock groups from Day 2?

```
set_clock_groups -asynchronous \
  -group [get_clocks ADC_Clk]         \
  -group [get_clocks Clk1]            \
  -group [get_clocks Clk2]            \
  -group [get_clocks {SRAM_CLK *altpll_0*}]
```

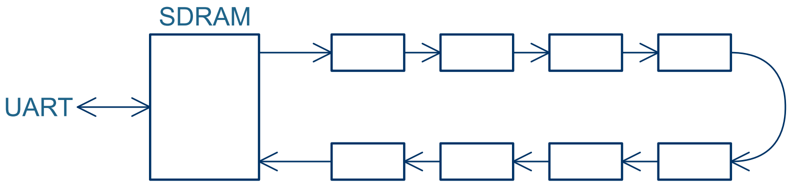
Outline

Sharing Resources



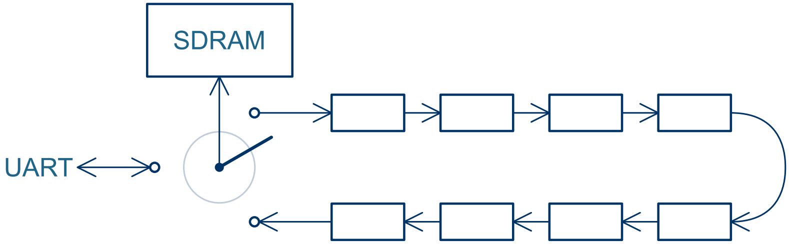
- Often many modules need access to the same resource, but only one module can interface with it at a time
- Examples include:
  - External memory (SDRAM, SD-card, etc.)
  - I^2C bus
  - etc.

Sharing Resources



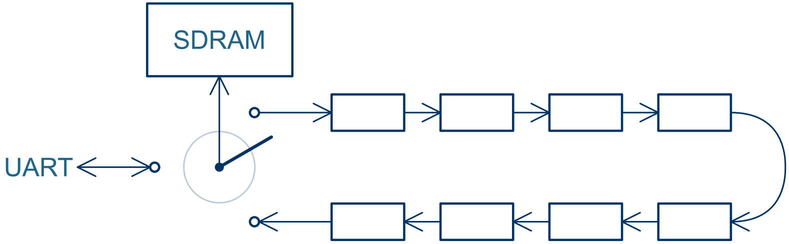
- Often many modules need access to the same resource, but only one module can interface with it at a time
- Examples include:
  - External memory (SDRAM, SD-card, etc.)
  - I^2C bus
  - etc.

Mutual Exclusion



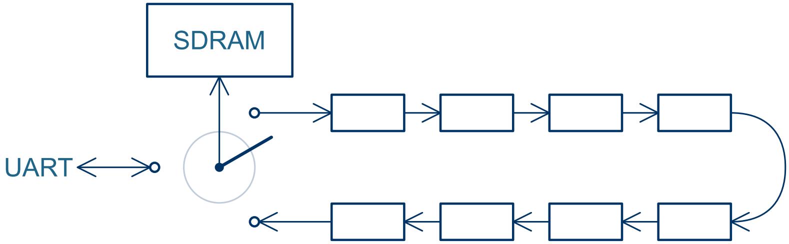
- Uses “Request” and “Grant” control lines
- The module would raise a request and then wait until it has been granted access before using the resource
- All the request lines go to a central control module that administers the granting of the resource

Mutual Exclusion



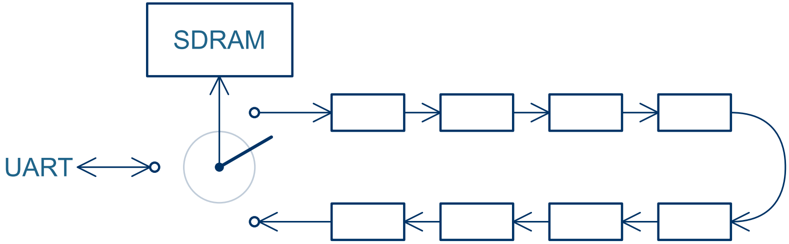
- Uses “Request” and “Grant” control lines
- The module would raise a request and then wait until it has been granted access before using the resource
- All the request lines go to a central control module that administers the granting of the resource

Mutual Exclusion



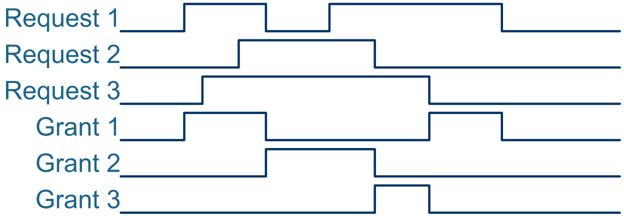
- Uses “Request” and “Grant” control lines
- The module would raise a request and then wait until it has been granted access before using the resource
- All the request lines go to a central control module that administers the granting of the resource

Mutual Exclusion



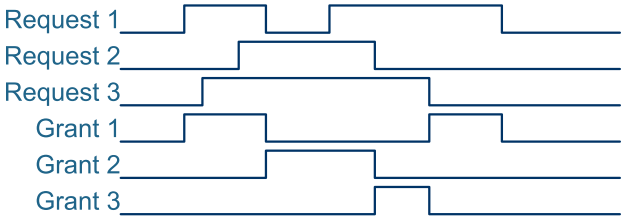
- Uses “Request” and “Grant” control lines
- The module would raise a request and then wait until it has been granted access before using the resource
- All the request lines go to a central control module that administers the granting of the resource

Round-robin Mutual Exclusion



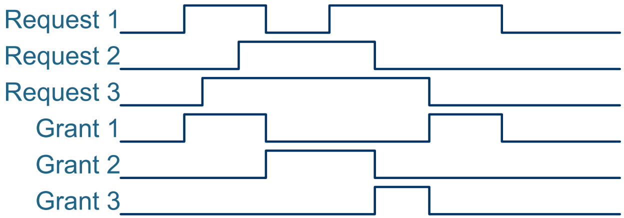
- The requests are serviced in a circular order
- Guaranteed no starvation
- Does not scale well (in the case of many modules, many clock-cycles must be wasted checking each module's request line)

Round-robin Mutual Exclusion



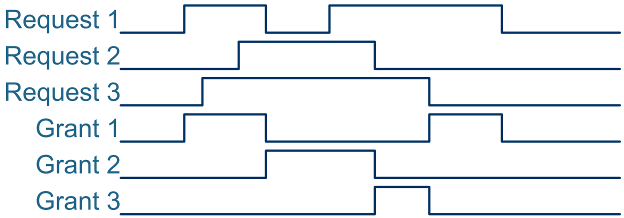
- The requests are serviced in a circular order
- Guaranteed no starvation
- Does not scale well (in the case of many modules, many clock-cycles must be wasted checking each module’s request line)

Round-robin Mutual Exclusion



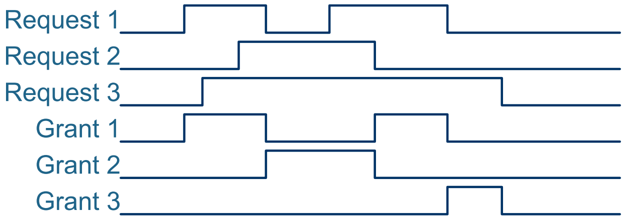
- The requests are serviced in a circular order
- Guaranteed no starvation
- Does not scale well (in the case of many modules, many clock-cycles must be wasted checking each module's request line)

Round-robin Mutual Exclusion



- The requests are serviced in a circular order
- Guaranteed no starvation
- Does not scale well (in the case of many modules, many clock-cycles must be wasted checking each module’s request line)

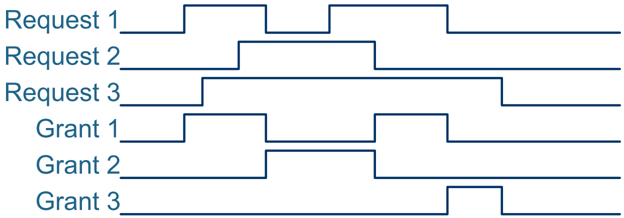
Priority-based Mutual Exclusion



- When more that one module requests access, the one with higher priority always receives the grant
- Can result in starvation of low-priority modules
- Fast (implementation can be a combinational circuit, providing a response within the same clock cycle)
- Scales well

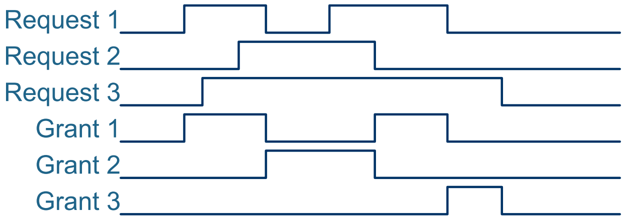
THE RADAR MASTERS COURSE

Priority-based Mutual Exclusion



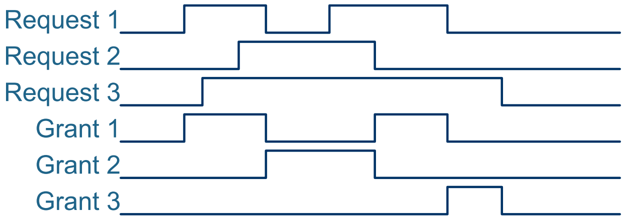
- When more that one module requests access, the one with higher priority always receives the grant
- Can result in starvation of low-priority modules
- Fast (implementation can be a combinational circuit, providing a response within the same clock cycle)
- Scales well

Priority-based Mutual Exclusion



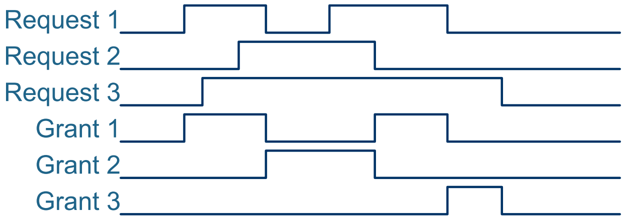
- When more that one module requests access, the one with higher priority always receives the grant
- Can result in starvation of low-priority modules
- Fast (implementation can be a combinational circuit, providing a response within the same clock cycle)
- Scales well

Priority-based Mutual Exclusion



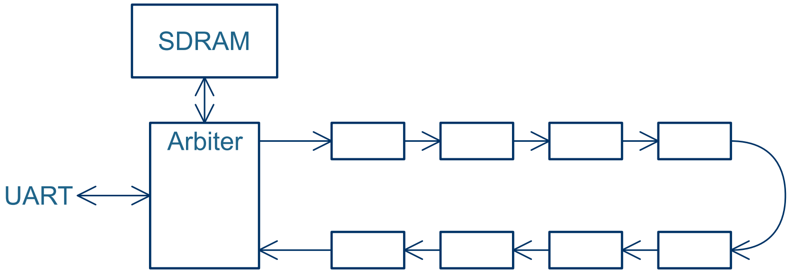
- When more that one module requests access, the one with higher priority always receives the grant
- Can result in starvation of low-priority modules
- Fast (implementation can be a combinational circuit, providing a response within the same clock cycle)
- Scales well

Priority-based Mutual Exclusion

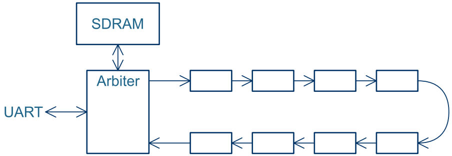


- When more that one module requests access, the one with higher priority always receives the grant
- Can result in starvation of low-priority modules
- Fast (implementation can be a combinational circuit, providing a response within the same clock cycle)
- Scales well

Arbitration

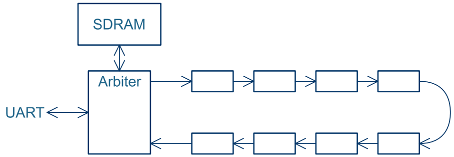


Arbitration



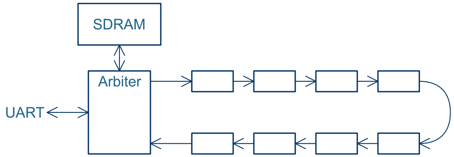
- The arbiter makes it look as if the resource has multiple independent interfaces. There are no control lines other than the interface itself.
- Often implemented by means of an embedded mutual exclusion unit (round-robin or priority based)
- The I^2C standard implements arbitration by collision-detection and random back-off: the modules monitor their own output, and when there is a mismatch, the module waits and tries again later.

Arbitration



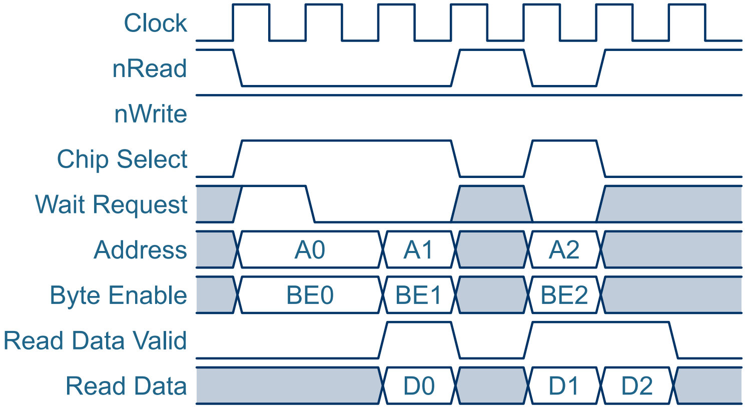
- The arbiter makes it look as if the resource has multiple independent interfaces. There are no control lines other than the interface itself.
- Often implemented by means of an embedded mutual exclusion unit (round-robin or priority based)
- The I^{2}C standard implements arbitration by collision-detection and random back-off: the modules monitor their own output, and when there is a mismatch, the module waits and tries again later.

Arbitration



- The arbiter makes it look as if the resource has multiple independent interfaces. There are no control lines other than the interface itself.
- Often implemented by means of an embedded mutual exclusion unit (round-robin or priority based)
- The I^{2}C standard implements arbitration by collision-detection and random back-off: the modules monitor their own output, and when there is a mismatch, the module waits and tries again later.

Avalon Bus Arbitration



Coffee Break...

Outline

Advanced Clocking

Clock Domains

Mutual Exclusion and Arbitration

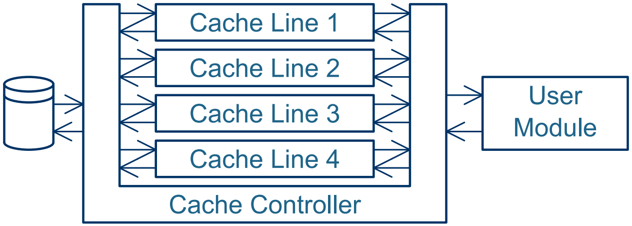
Caching Systems

Flow Control

Practical – NCO and DDC

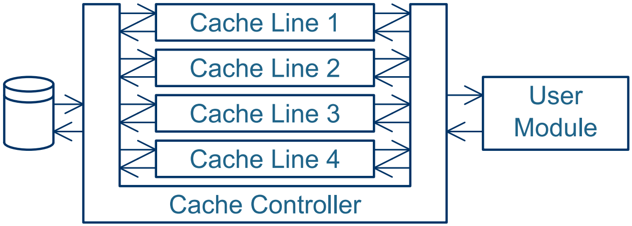
Numerically-controlled Oscillator

General Cache



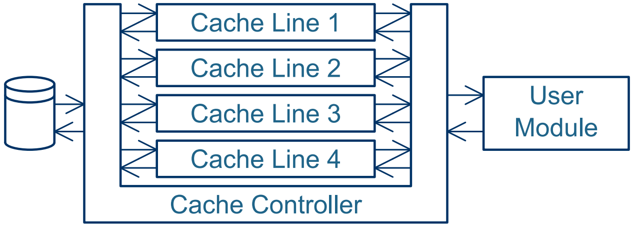
- Lines are typically one page in size (4 kiB in modern PCs, typically smaller on embedded systems)
- The controller does address translation between the external and cache line addresses
- Read is on-demand, flushing the least recently used line and loading the content into it
- Controller blocks the user module when a cache miss occurs

General Cache



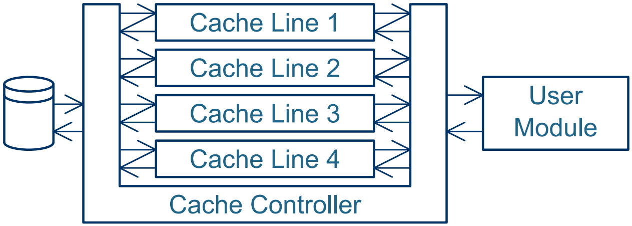
- Lines are typically one page in size (4 kiB in modern PCs, typically smaller on embedded systems)
- The controller does address translation between the external and cache line addresses
- Read is on-demand, flushing the least recently used line and loading the content into it
- Controller blocks the user module when a cache miss occurs

General Cache



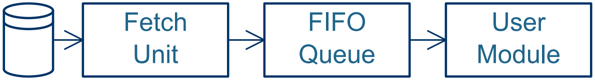
- Lines are typically one page in size (4 kiB in modern PCs, typically smaller on embedded systems)
- The controller does address translation between the external and cache line addresses
- Read is on-demand, flushing the least recently used line and loading the content into it
- Controller blocks the user module when a cache miss occurs

General Cache



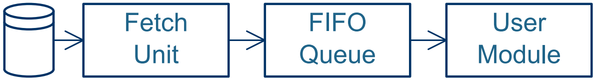
- Lines are typically one page in size (4 kiB in modern PCs, typically smaller on embedded systems)
- The controller does address translation between the external and cache line addresses
- Read is on-demand, flushing the least recently used line and loading the content into it
- Controller blocks the user module when a cache miss occurs

Pre-fetch Read Cache



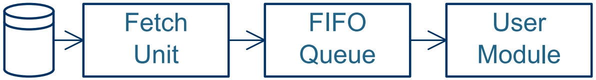
- Reading is continuous, and the user module is only blocked when the queue is empty
- The queue is typically multiple pages in size
- New data is fetched as soon as there is one page open on the queue

Pre-fetch Read Cache



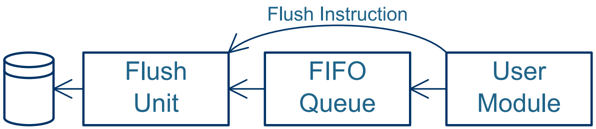
- Reading is continuous, and the user module is only blocked when the queue is empty
- The queue is typically multiple pages in size
- New data is fetched as soon as there is one page open on the queue

Pre-fetch Read Cache



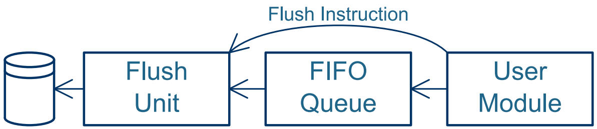
- Reading is continuous, and the user module is only blocked when the queue is empty
- The queue is typically multiple pages in size
- New data is fetched as soon as there is one page open on the queue

Write Cache



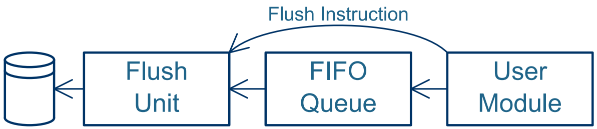
- Writing is continuous, and the user module is only blocked when the queue is full
- Alternatively, the user module is never blocked, and data is lost when the queue is full
- The flush unit automatically flushes the queue when there is one page of data in it
- The user module can also manually flush the queue if required

Write Cache



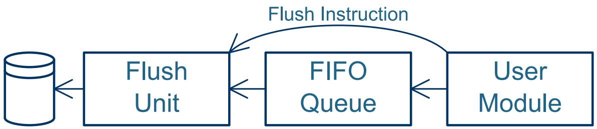
- Writing is continuous, and the user module is only blocked when the queue is full
- Alternatively, the user module is never blocked, and data is lost when the queue is full
- The flush unit automatically flushes the queue when there is one page of data in it
- The user module can also manually flush the queue if required

Write Cache



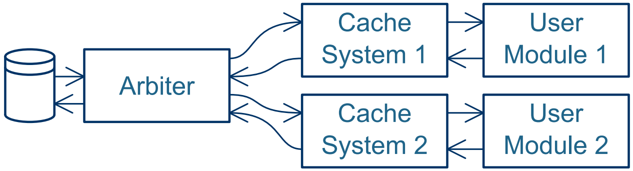
- Writing is continuous, and the user module is only blocked when the queue is full
- Alternatively, the user module is never blocked, and data is lost when the queue is full
- The flush unit automatically flushes the queue when there is one page of data in it
- The user module can also manually flush the queue if required

Write Cache



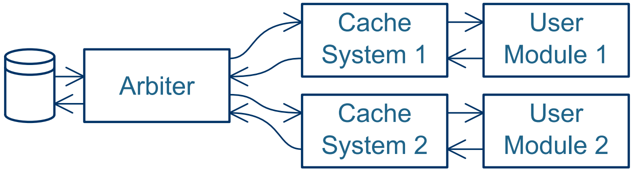
- Writing is continuous, and the user module is only blocked when the queue is full
- Alternatively, the user module is never blocked, and data is lost when the queue is full
- The flush unit automatically flushes the queue when there is one page of data in it
- The user module can also manually flush the queue if required

Cache Coherence



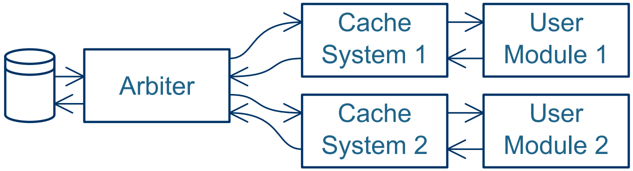
- Two independent cache system can potentially refer to the same place in external storage
- When the one user module reads what the other is writing, the two cache system must remain up to date with each other
- When the two user modules write to different places in the same page, a flush must not overwrite the other's data
- Many schemes exist to mitigate these and other issues, which are outside the scope of this course

Cache Coherence



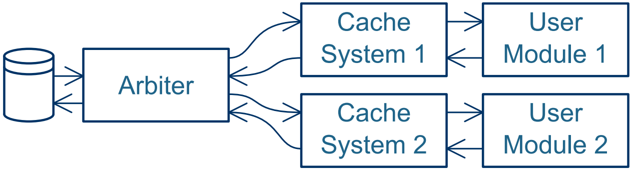
- Two independent cache system can potentially refer to the same place in external storage
- When the one user module reads what the other is writing, the two cache system must remain up to date with each other
- When the two user modules write to different places in the same page, a flush must not overwrite the other's data
- Many schemes exist to mitigate these and other issues, which are outside the scope of this course

Cache Coherence



- Two independent cache system can potentially refer to the same place in external storage
- When the one user module reads what the other is writing, the two cache system must remain up to date with each other
- When the two user modules write to different places in the same page, a flush must not overwrite the other's data
- Many schemes exist to mitigate these and other issues, which are outside the scope of this course

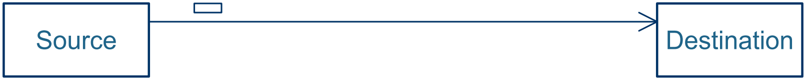
Cache Coherence



- Two independent cache system can potentially refer to the same place in external storage
- When the one user module reads what the other is writing, the two cache system must remain up to date with each other
- When the two user modules write to different places in the same page, a flush must not overwrite the other's data
- Many schemes exist to mitigate these and other issues, which are outside the scope of this course

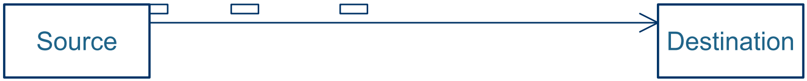
Outline

Fire and Forget



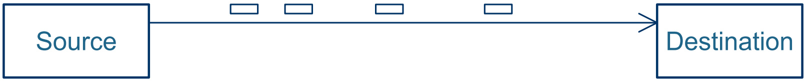
- No feedback from the destination to the source
- Assumes that the destination will handle the data, or
- The application can accept data loss

Fire and Forget



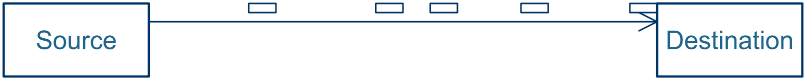
- No feedback from the destination to the source
- Assumes that the destination will handle the data, or
- The application can accept data loss

Fire and Forget



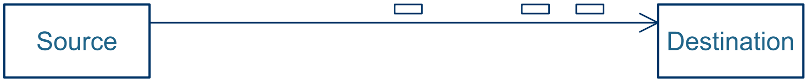
- No feedback from the destination to the source
- Assumes that the destination will handle the data, or
- The application can accept data loss

Fire and Forget



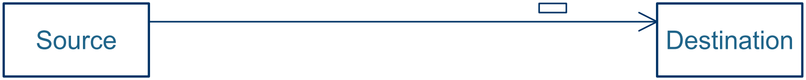
- No feedback from the destination to the source
- Assumes that the destination will handle the data, or
- The application can accept data loss

Fire and Forget



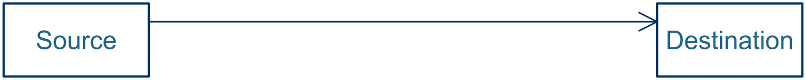
- No feedback from the destination to the source
- Assumes that the destination will handle the data, or
- The application can accept data loss

Fire and Forget



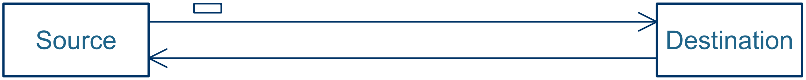
- No feedback from the destination to the source
- Assumes that the destination will handle the data, or
- The application can accept data loss

Fire and Forget



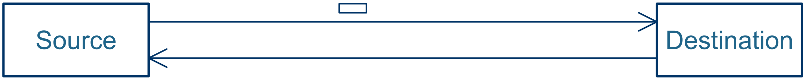
- No feedback from the destination to the source
- Assumes that the destination will handle the data, or
- The application can accept data loss

Command-Response



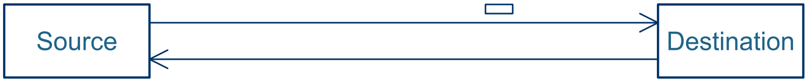
- The source sends one packet at a time
- And waits for the response from the destination

Command-Response



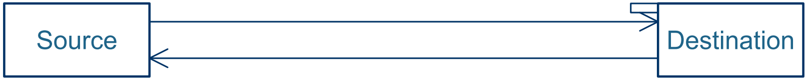
- The source sends one packet at a time
- And waits for the response from the destination

Command-Response



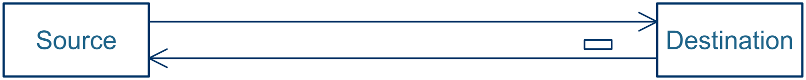
- The source sends one packet at a time
- And waits for the response from the destination

Command-Response



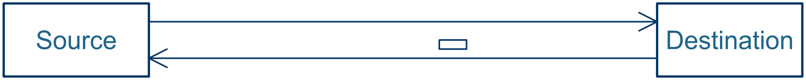
- The source sends one packet at a time
- And waits for the response from the destination

Command-Response



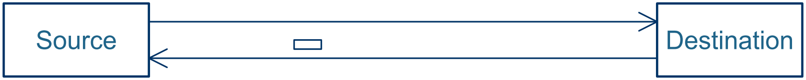
- The source sends one packet at a time
- And waits for the response from the destination

Command-Response



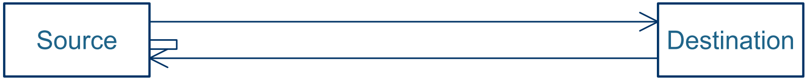
- The source sends one packet at a time
- And waits for the response from the destination

Command-Response



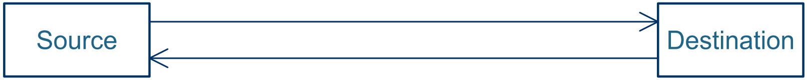
- The source sends one packet at a time
- And waits for the response from the destination

Command-Response



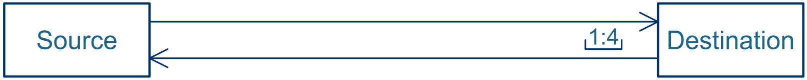
- The source sends one packet at a time
- And waits for the response from the destination

Command-Response



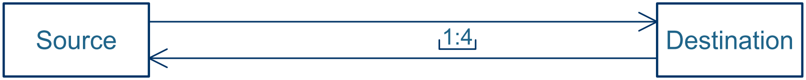
- The source sends one packet at a time
- And waits for the response from the destination

Sliding Window



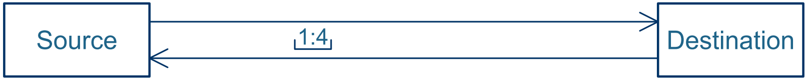
- The destination advertises a window
- The source aims to fill the window with data
- This is used in TCP, among other protocols

Sliding Window



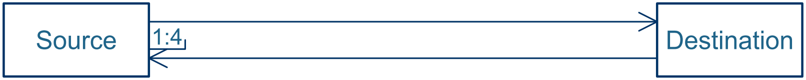
- The destination advertises a window
- The source aims to fill the window with data
- This is used in TCP, among other protocols

Sliding Window



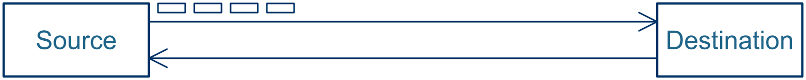
- The destination advertises a window
- The source aims to fill the window with data
- This is used in TCP, among other protocols

Sliding Window



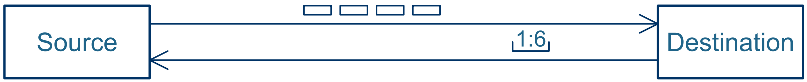
- The destination advertises a window
- The source aims to fill the window with data
- This is used in TCP, among other protocols

Sliding Window



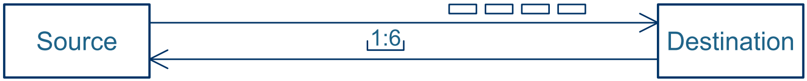
- The destination advertises a window
- The source aims to fill the window with data
- This is used in TCP, among other protocols

Sliding Window



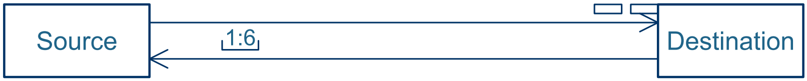
- The destination advertises a window
- The source aims to fill the window with data
- This is used in TCP, among other protocols

Sliding Window



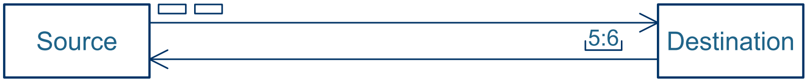
- The destination advertises a window
- The source aims to fill the window with data
- This is used in TCP, among other protocols

Sliding Window



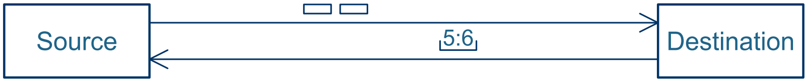
- The destination advertises a window
- The source aims to fill the window with data
- This is used in TCP, among other protocols

Sliding Window



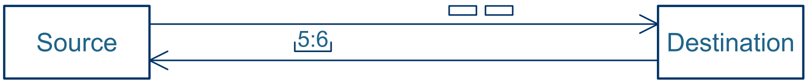
- The destination advertises a window
- The source aims to fill the window with data
- This is used in TCP, among other protocols

Sliding Window



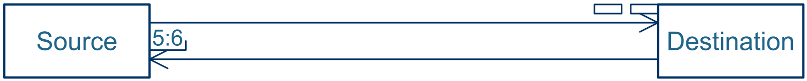
- The destination advertises a window
- The source aims to fill the window with data
- This is used in TCP, among other protocols

Sliding Window



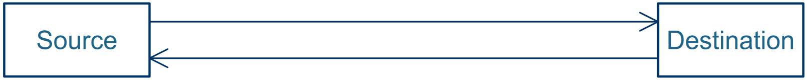
- The destination advertises a window
- The source aims to fill the window with data
- This is used in TCP, among other protocols

Sliding Window



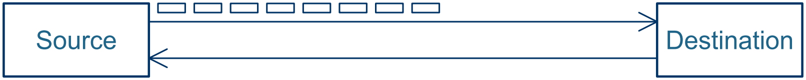
- The destination advertises a window
- The source aims to fill the window with data
- This is used in TCP, among other protocols

Sliding Window



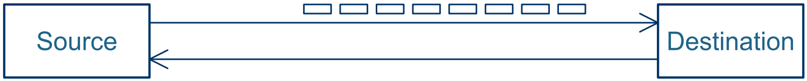
- The destination advertises a window
- The source aims to fill the window with data
- This is used in TCP, among other protocols

Credit Based (1)



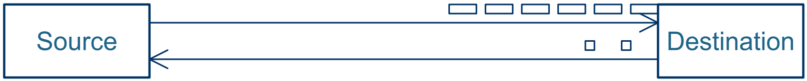
- The source assumes that the destination has a certain space
- The destination acknowledges all incoming packets
- The source sends packets aiming to keep the credit on zero
- Works best when the credit is large enough to keep both directions full at all times

Credit Based (1)



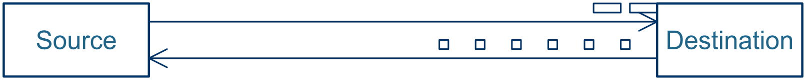
- The source assumes that the destination has a certain space
- The destination acknowledges all incoming packets
- The source sends packets aiming to keep the credit on zero
- Works best when the credit is large enough to keep both directions full at all times

Credit Based (1)



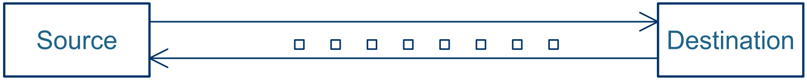
- The source assumes that the destination has a certain space
- The destination acknowledges all incoming packets
- The source sends packets aiming to keep the credit on zero
- Works best when the credit is large enough to keep both directions full at all times

Credit Based (1)



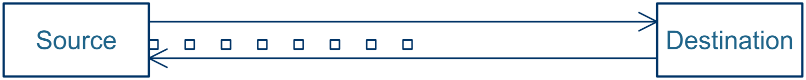
- The source assumes that the destination has a certain space
- The destination acknowledges all incoming packets
- The source sends packets aiming to keep the credit on zero
- Works best when the credit is large enough to keep both directions full at all times

Credit Based (1)



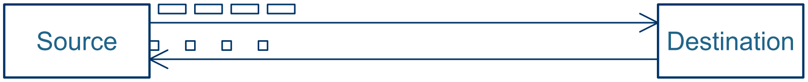
- The source assumes that the destination has a certain space
- The destination acknowledges all incoming packets
- The source sends packets aiming to keep the credit on zero
- Works best when the credit is large enough to keep both directions full at all times

Credit Based (1)



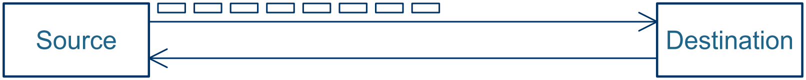
- The source assumes that the destination has a certain space
- The destination acknowledges all incoming packets
- The source sends packets aiming to keep the credit on zero
- Works best when the credit is large enough to keep both directions full at all times

Credit Based (1)



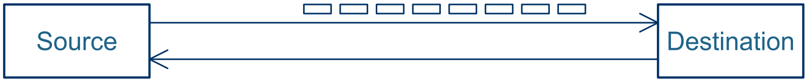
- The source assumes that the destination has a certain space
- The destination acknowledges all incoming packets
- The source sends packets aiming to keep the credit on zero
- Works best when the credit is large enough to keep both directions full at all times

Credit Based (1)



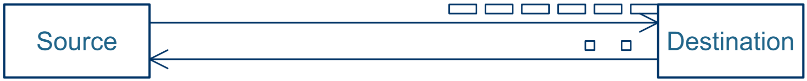
- The source assumes that the destination has a certain space
- The destination acknowledges all incoming packets
- The source sends packets aiming to keep the credit on zero
- Works best when the credit is large enough to keep both directions full at all times

Credit Based (1)



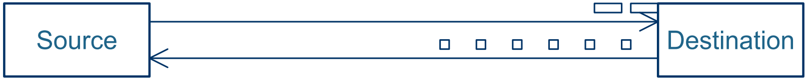
- The source assumes that the destination has a certain space
- The destination acknowledges all incoming packets
- The source sends packets aiming to keep the credit on zero
- Works best when the credit is large enough to keep both directions full at all times

Credit Based (1)



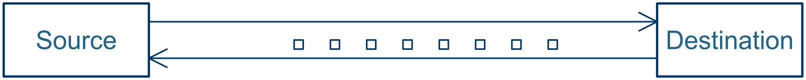
- The source assumes that the destination has a certain space
- The destination acknowledges all incoming packets
- The source sends packets aiming to keep the credit on zero
- Works best when the credit is large enough to keep both directions full at all times

Credit Based (1)



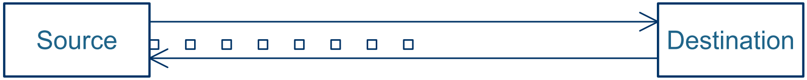
- The source assumes that the destination has a certain space
- The destination acknowledges all incoming packets
- The source sends packets aiming to keep the credit on zero
- Works best when the credit is large enough to keep both directions full at all times

Credit Based (1)



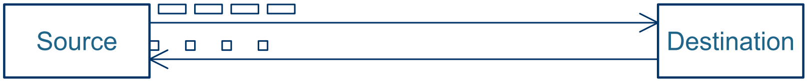
- The source assumes that the destination has a certain space
- The destination acknowledges all incoming packets
- The source sends packets aiming to keep the credit on zero
- Works best when the credit is large enough to keep both directions full at all times

Credit Based (1)



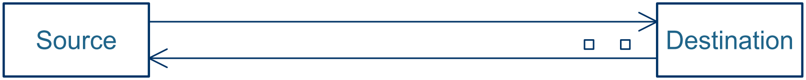
- The source assumes that the destination has a certain space
- The destination acknowledges all incoming packets
- The source sends packets aiming to keep the credit on zero
- Works best when the credit is large enough to keep both directions full at all times

Credit Based (1)



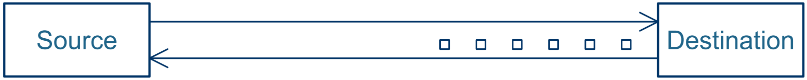
- The source assumes that the destination has a certain space
- The destination acknowledges all incoming packets
- The source sends packets aiming to keep the credit on zero
- Works best when the credit is large enough to keep both directions full at all times

Credit Based (2)



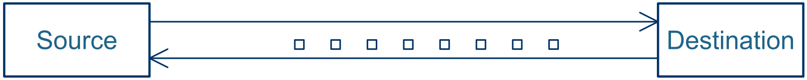
- Instead of the source starting with credit, an alternative algorithm lets the destination provide credit to the source
- This is used in PCIe, among other protocols

Credit Based (2)



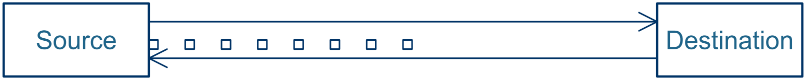
- Instead of the source starting with credit, an alternative algorithm lets the destination provide credit to the source
- This is used in PCIe, among other protocols

Credit Based (2)



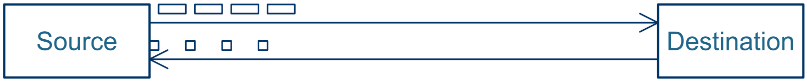
- Instead of the source starting with credit, an alternative algorithm lets the destination provide credit to the source
- This is used in PCIe, among other protocols

Credit Based (2)



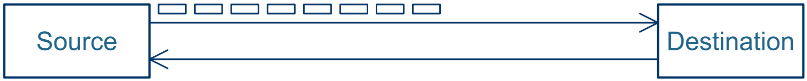
- Instead of the source starting with credit, an alternative algorithm lets the destination provide credit to the source
- This is used in PCIe, among other protocols

Credit Based (2)



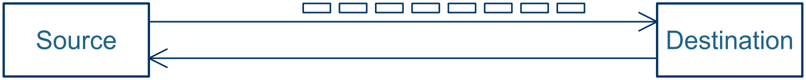
- Instead of the source starting with credit, an alternative algorithm lets the destination provide credit to the source
- This is used in PCIe, among other protocols

Credit Based (2)



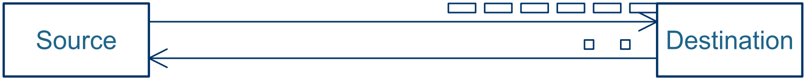
- Instead of the source starting with credit, an alternative algorithm lets the destination provide credit to the source
- This is used in PCIe, among other protocols

Credit Based (2)



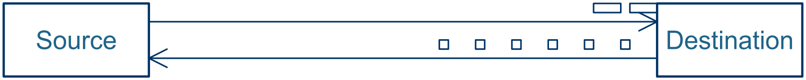
- Instead of the source starting with credit, an alternative algorithm lets the destination provide credit to the source
- This is used in PCIe, among other protocols

Credit Based (2)



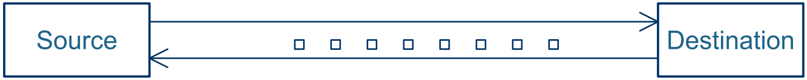
- Instead of the source starting with credit, an alternative algorithm lets the destination provide credit to the source
- This is used in PCIe, among other protocols

Credit Based (2)



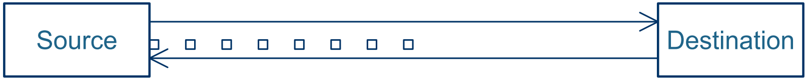
- Instead of the source starting with credit, an alternative algorithm lets the destination provide credit to the source
- This is used in PCIe, among other protocols

Credit Based (2)



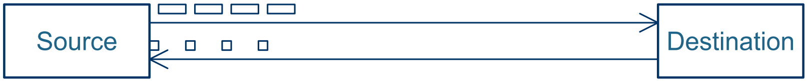
- Instead of the source starting with credit, an alternative algorithm lets the destination provide credit to the source
- This is used in PCIe, among other protocols

Credit Based (2)



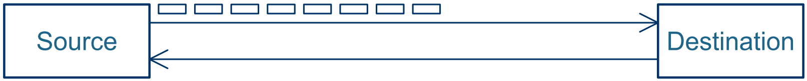
- Instead of the source starting with credit, an alternative algorithm lets the destination provide credit to the source
- This is used in PCIe, among other protocols

Credit Based (2)



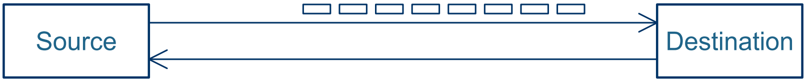
- Instead of the source starting with credit, an alternative algorithm lets the destination provide credit to the source
- This is used in PCIe, among other protocols

Credit Based (2)



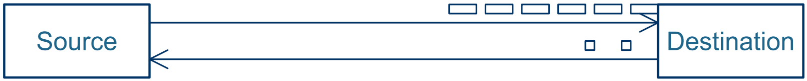
- Instead of the source starting with credit, an alternative algorithm lets the destination provide credit to the source
- This is used in PCIe, among other protocols

Credit Based (2)



- Instead of the source starting with credit, an alternative algorithm lets the destination provide credit to the source
- This is used in PCIe, among other protocols

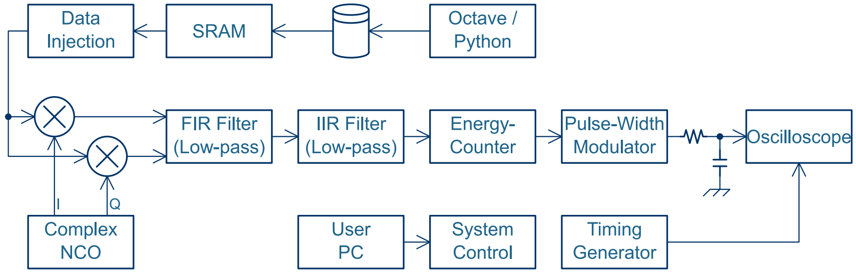
Credit Based (2)



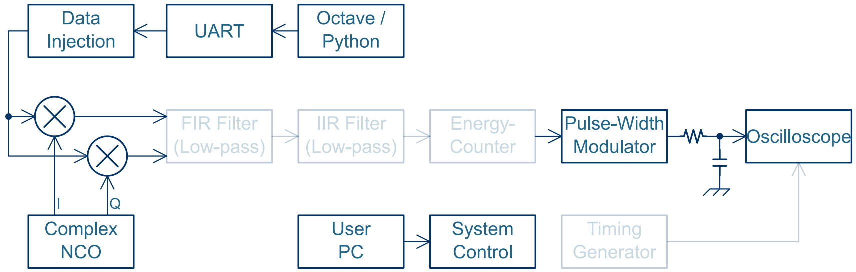
- Instead of the source starting with credit, an alternative algorithm lets the destination provide credit to the source
- This is used in PCIe, among other protocols

Outline

Practical – NCO and DDC



Practical – NCO and DDC



Outline

Advanced Clocking

Clock Domains

Mutual Exclusion and Arbitration

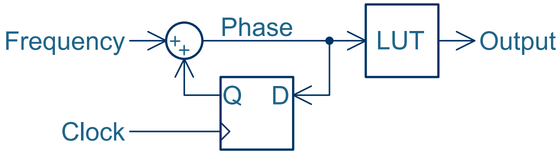
Caching Systems

Flow Control

Practical – NCO and DDC

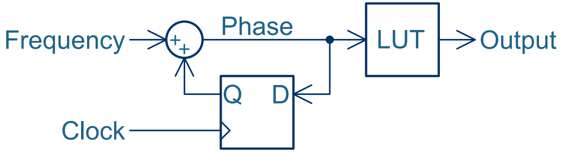
Numerically-controlled Oscillator

Numerically-controlled Oscillator



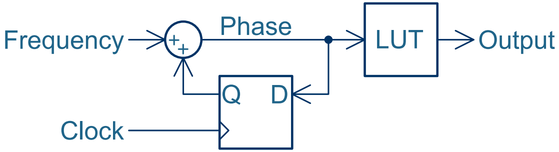
- Integrate frequency to obtain phase
- Use on-chip memory blocks as a look-up table to convert phase to the intended waveform
- Use dual-port ROM to obtain sine and cosine from the same LUT
- Higher resolution can be obtained with a CORDIC-based sine-cosine module

Numerically-controlled Oscillator



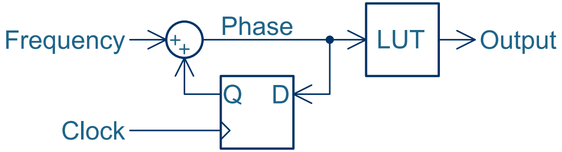
- Integrate frequency to obtain phase
- Use on-chip memory blocks as a look-up table to convert phase to the intended waveform
- Use dual-port ROM to obtain sine and cosine from the same LUT
- Higher resolution can be obtained with a CORDIC-based sine-cosine module

Numerically-controlled Oscillator



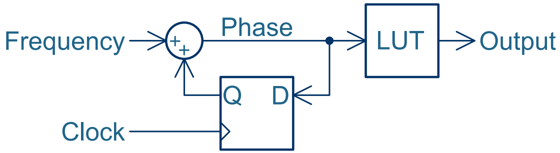
- Integrate frequency to obtain phase
- Use on-chip memory blocks as a look-up table to convert phase to the intended waveform
- Use dual-port ROM to obtain sine and cosine from the same LUT
- Higher resolution can be obtained with a CORDIC-based sine-cosine module

Numerically-controlled Oscillator



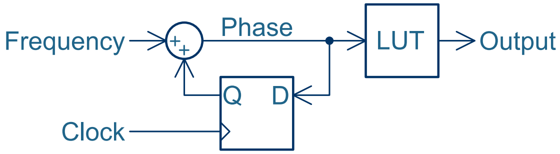
- Integrate frequency to obtain phase
- Use on-chip memory blocks as a look-up table to convert phase to the intended waveform
- Use dual-port ROM to obtain sine and cosine from the same LUT
- Higher resolution can be obtained with a CORDIC-based sine-cosine module

Numerically-controlled Oscillator



- Integrate frequency to obtain phase
- Use on-chip memory blocks as a look-up table to convert phase to the intended waveform
- Use dual-port ROM to obtain sine and cosine from the same LUT
- Higher resolution can be obtained with a CORDIC-based sine-cosine module

Numerically-controlled Oscillator



$$f_{out} = f_s \cdot \frac{f_{in}}{2^N}, \quad N = 32, \quad f_s = 50 \text{ MHz}$$

Select References

- Stephen Brown and Zvonko Vranesic
  Fundamentals of Digital Logic with Verilog Design, 2nd Edition
  ISBN 978-0-07-721164-6
- Merrill L Skolnik
  Introduction to RADAR Systems
  ISBN 978-0-07-288138-7
- Mark A. Richards and James A. Scheer
  Principles of Modern Radar: Basic Principles
  ISBN 978-1-89-112152-4
- Deepak Kumar Tala
  World of ASIC
  `http://www.asic-world.com/`
- Jean P. Nicolle
  FPGA 4 Fun
  `http://www.fpga4fun.com/`

FPGA Development for Radar, Radio-Astronomy and Communications

THE RADAR MASTERS COURSE

Dept. Electrical Engineering, University of Cape Town
Private Bag, Rondebosch, 7701, South Africa

http://www.rrsg.uct.ac.za

Presented by John-Philip Taylor

Convened by Dr Stephen Paine

Day 4 – 2 May 2022