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 Prof Daniel O'Hagan

Tutored by Stephen Paine and Randy Cheng

Day 4 – 20 July 2017

Outline

Advanced Clocking

Clock Domains

Practical

Outline

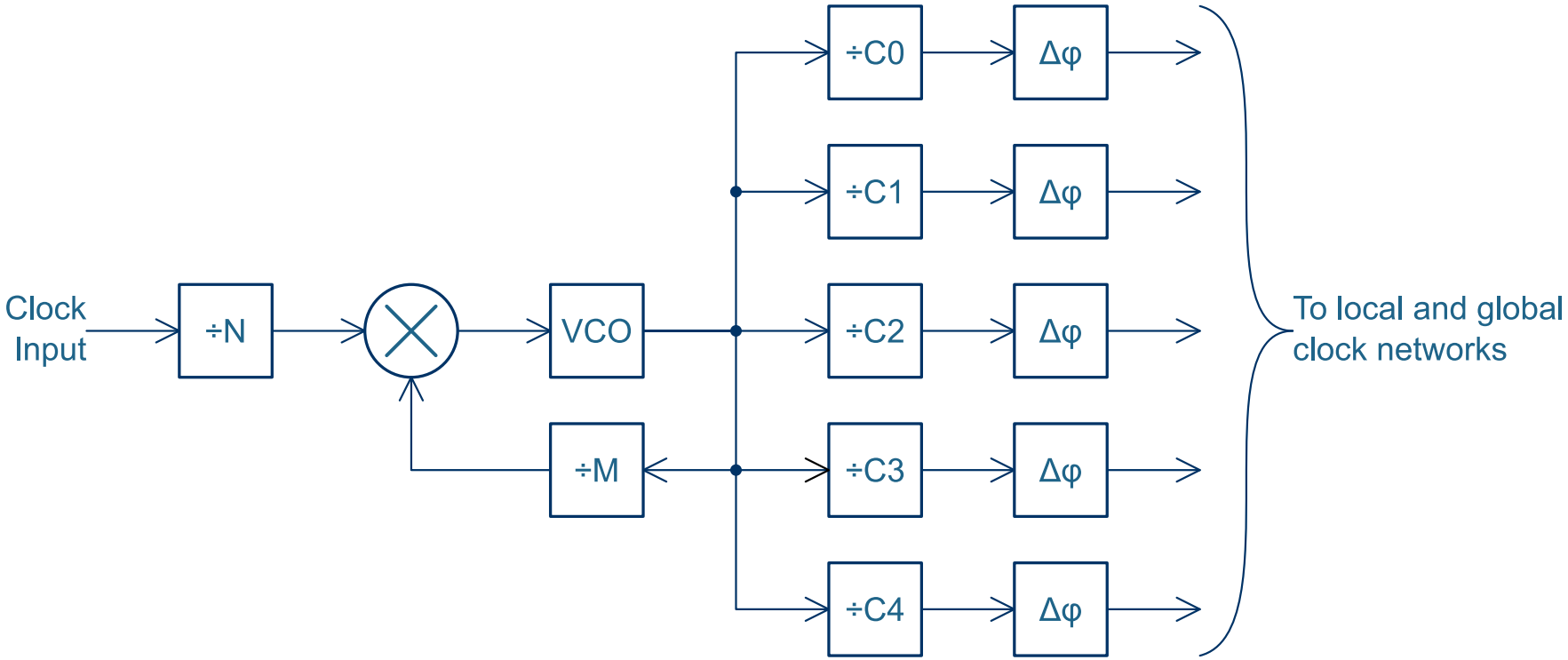
Advanced Clocking

Clock Domains

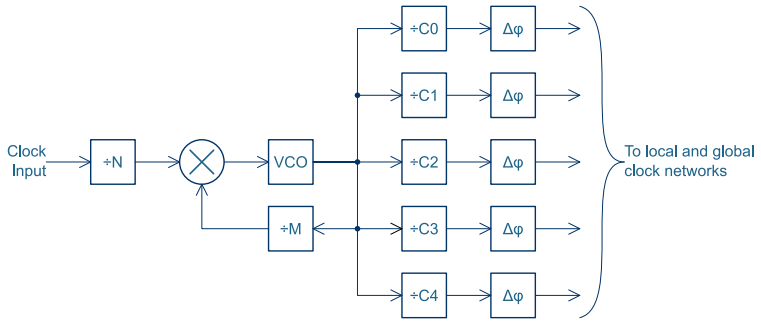
Practical

THE RADAR MASTERS COURSE

Phase-locked Loop

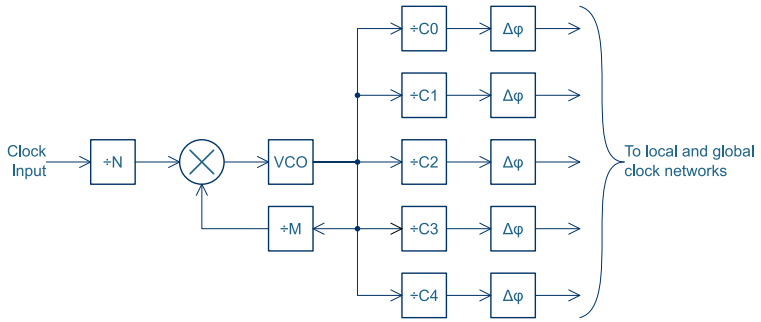


Phase-locked Loop



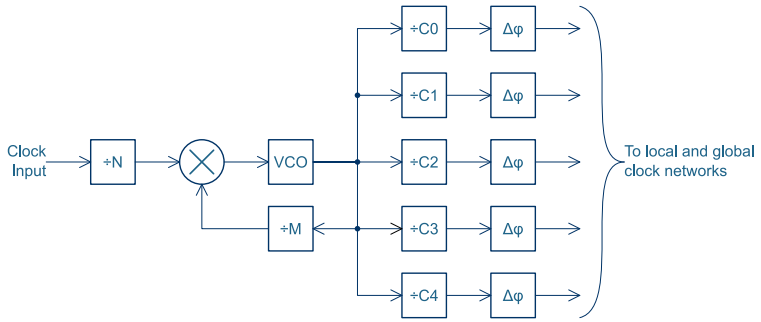
- Useful for generating a range of related clocks
- Output clock rising and falling edges can be set, independently, to a resolution equal to the VCO period, which is typically about 1 ns $\pm$ 500 ps
- All output clocks can be considered to be in the same clock domain
- Noise sensitive applications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

Phase-locked Loop



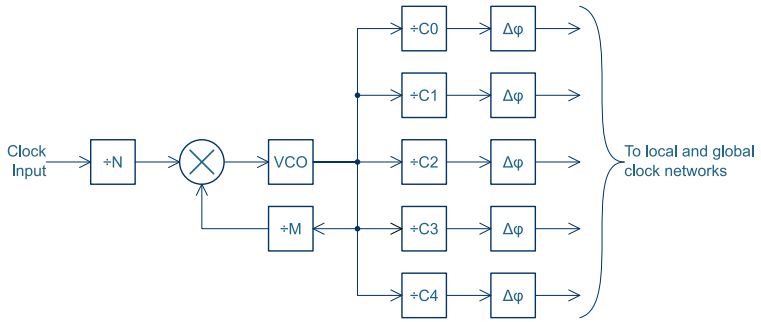
- Useful for generating a range of related clocks
- Output clock rising and falling edges can be set, independently, to a resolution equal to the VCO period, which is typically about 1 ns $\pm$ 500 ps
- All output clocks can be considered to be in the same clock domain
- Noise sensitive applications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

Phase-locked Loop



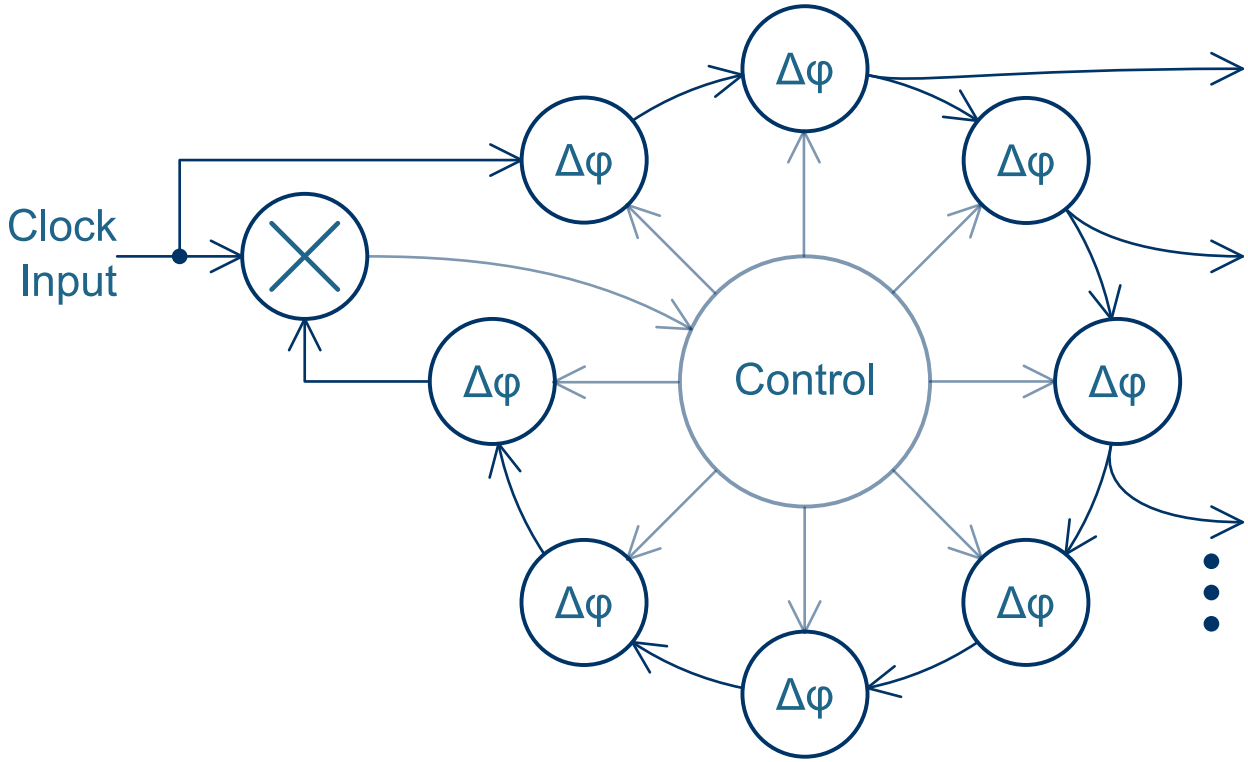
- Useful for generating a range of related clocks
- Output clock rising and falling edges can be set, independently, to a resolution equal to the VCO period, which is typically about 1 ns $\pm$ 500 ps
- All output clocks can be considered to be in the same clock domain
- Noise sensitive applications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

Phase-locked Loop

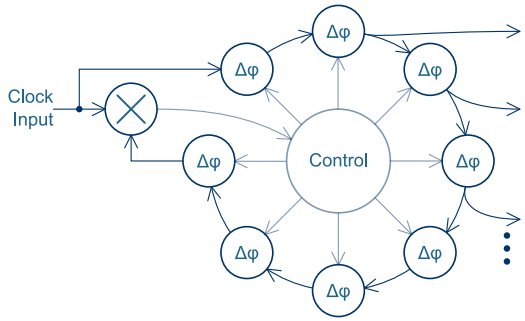


- Useful for generating a range of related clocks
- Output clock rising and falling edges can be set, independently, to a resolution equal to the VCO period, which is typically about 1 ns $\pm$ 500 ps
- All output clocks can be considered to be in the same clock domain
- Noise sensitive applications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

Delay-locked Loop

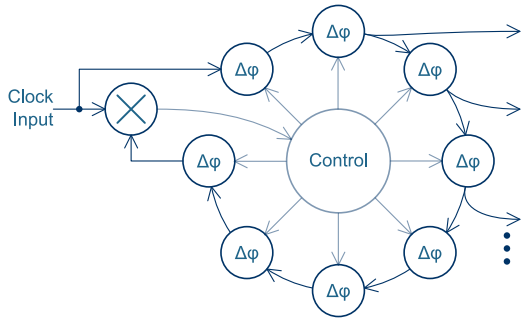


Delay-locked Loop



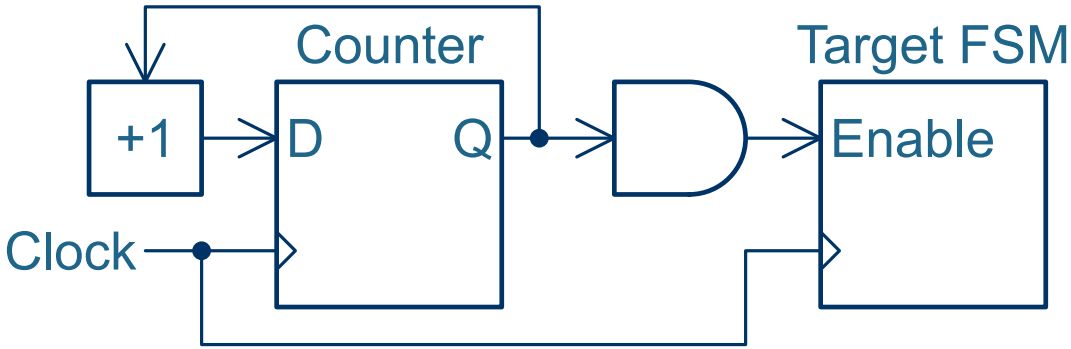
- Useful for performing phase compensation when interfacing to fast peripherals
- All output clocks can be considered to be in the same clock domain as the input clock

Delay-locked Loop

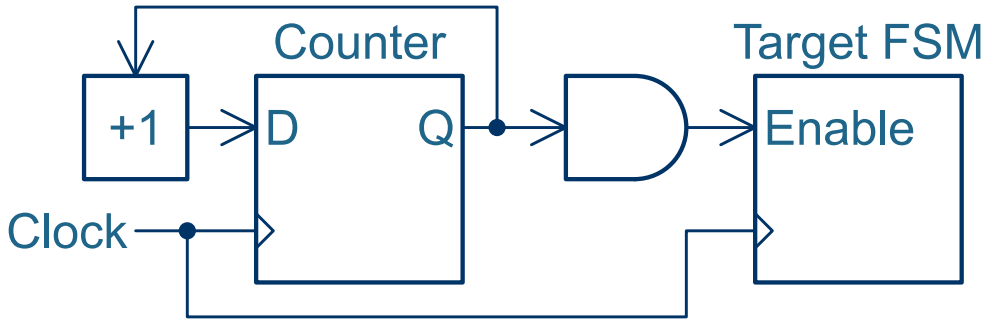


- Useful for performing phase compensation when interfacing to fast peripherals
- All output clocks can be considered to be in the same clock domain as the input clock

Clock-Enable Generated

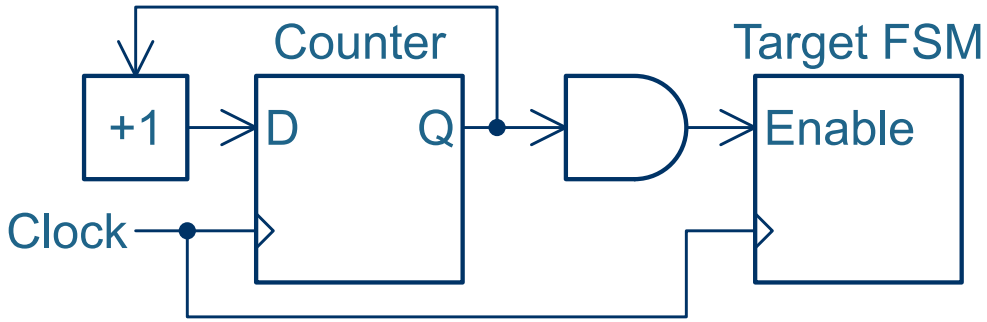


Clock-Enable Generated



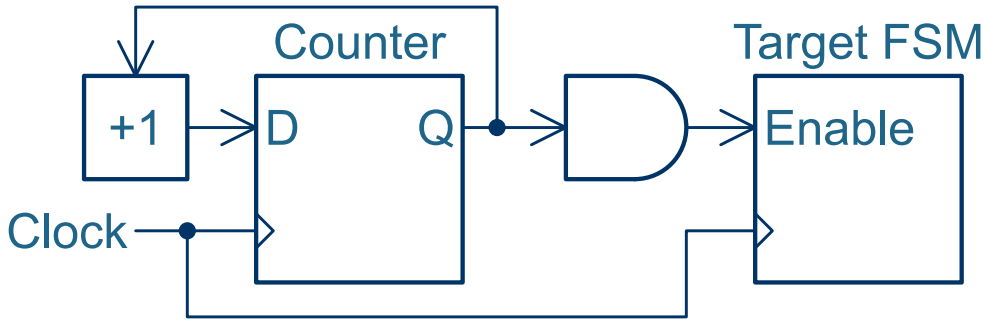
- Useful when mixing fast and slow systems
- Potential to reduce overall device power usage
- Cannot be used for clocking external devices
- The multi-cycle timing requirements must be manually specified

Clock-Enable Generated



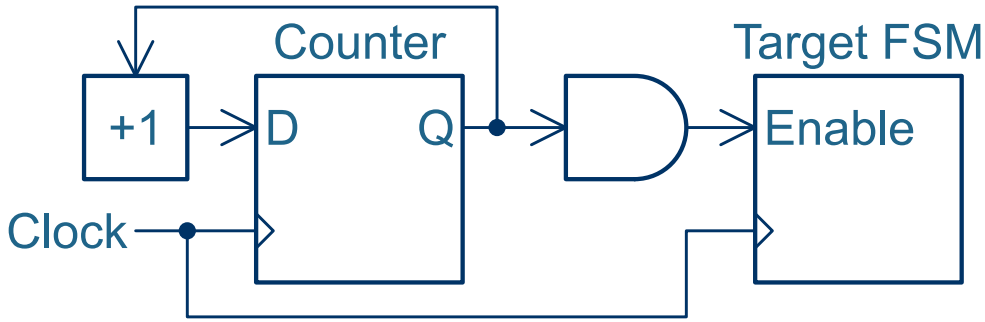
- Useful when mixing fast and slow systems
- Potential to reduce overall device power usage
- Cannot be used for clocking external devices
- The multi-cycle timing requirements must be manually specified

Clock-Enable Generated



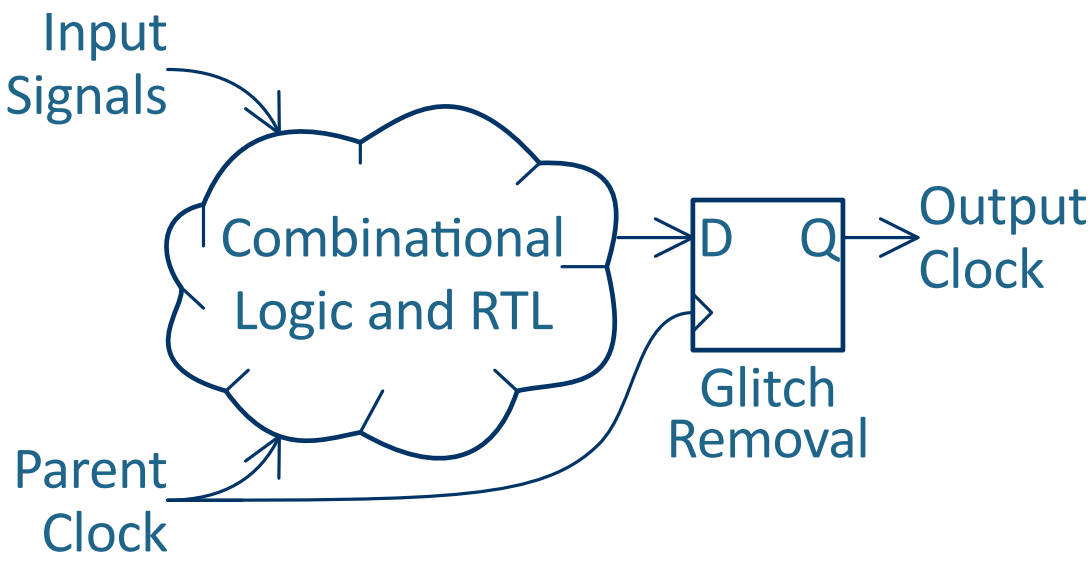
- Useful when mixing fast and slow systems
- Potential to reduce overall device power usage
- Cannot be used for clocking external devices
- The multi-cycle timing requirements must be manually specified

Clock-Enable Generated

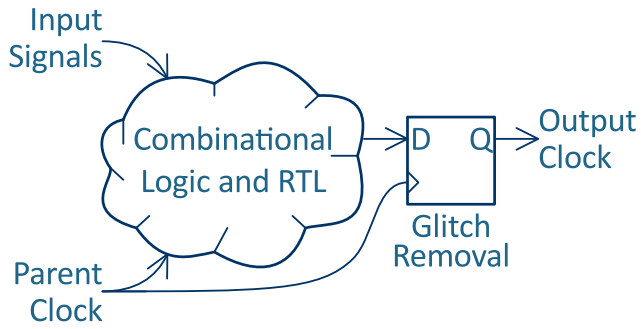


- Useful when mixing fast and slow systems
- Potential to reduce overall device power usage
- Cannot be used for clocking external devices
- The multi-cycle timing requirements must be manually specified

Ripple and Gated Clocks

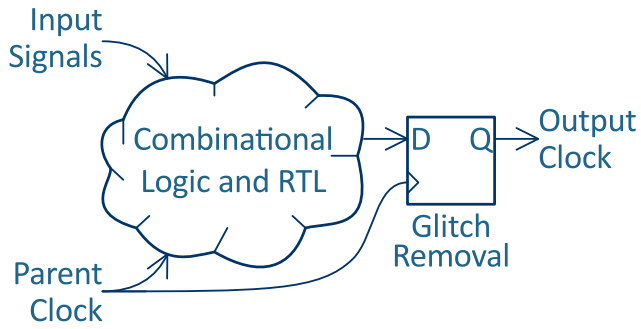


Ripple and Gated Clocks



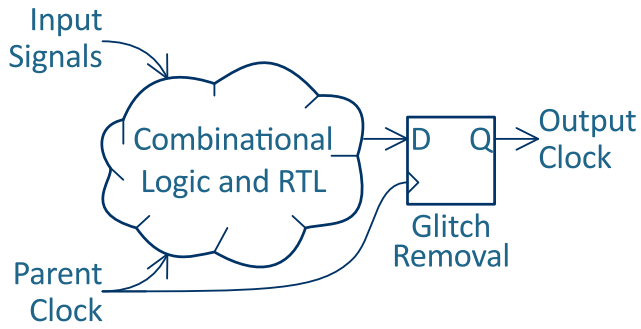
- Unrelated to all other clocks in the system, including the parent clock
- Often necessitates complex clock-domain crossing schemes to the general system clock
- Useful when:
  - Requiring very low power usage
  - The frequency must change often
  - The clock must drive an external device
  - etc.

Ripple and Gated Clocks



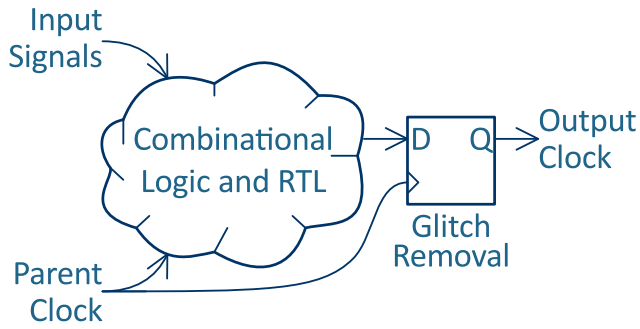
- Unrelated to all other clocks in the system, including the parent clock
- Often necessitates complex clock-domain crossing schemes to the general system clock
- Useful when:
  - Requiring very low power usage
  - The frequency must change often
  - The clock must drive an external device
  - etc.

Ripple and Gated Clocks



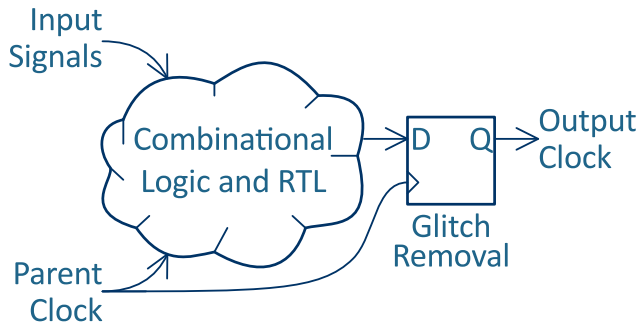
- Unrelated to all other clocks in the system, including the parent clock
- Often necessitates complex clock-domain crossing schemes to the general system clock
- Useful when:
  - Requiring very low power usage
  - The frequency must change often
  - The clock must drive an external device
  - etc.

Ripple and Gated Clocks



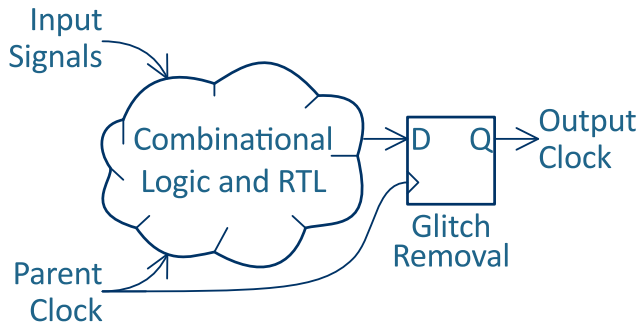
- Unrelated to all other clocks in the system, including the parent clock
- Often necessitates complex clock-domain crossing schemes to the general system clock
- Useful when:
  - Requiring very low power usage
  - The frequency must change often
  - The clock must drive an external device
  - etc.

Ripple and Gated Clocks



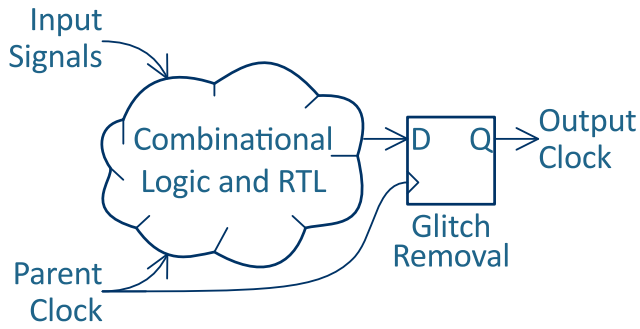
- Unrelated to all other clocks in the system, including the parent clock
- Often necessitates complex clock-domain crossing schemes to the general system clock
- Useful when:
  - Requiring very low power usage
  - The frequency must change often
  - The clock must drive an external device
  - etc.

Ripple and Gated Clocks



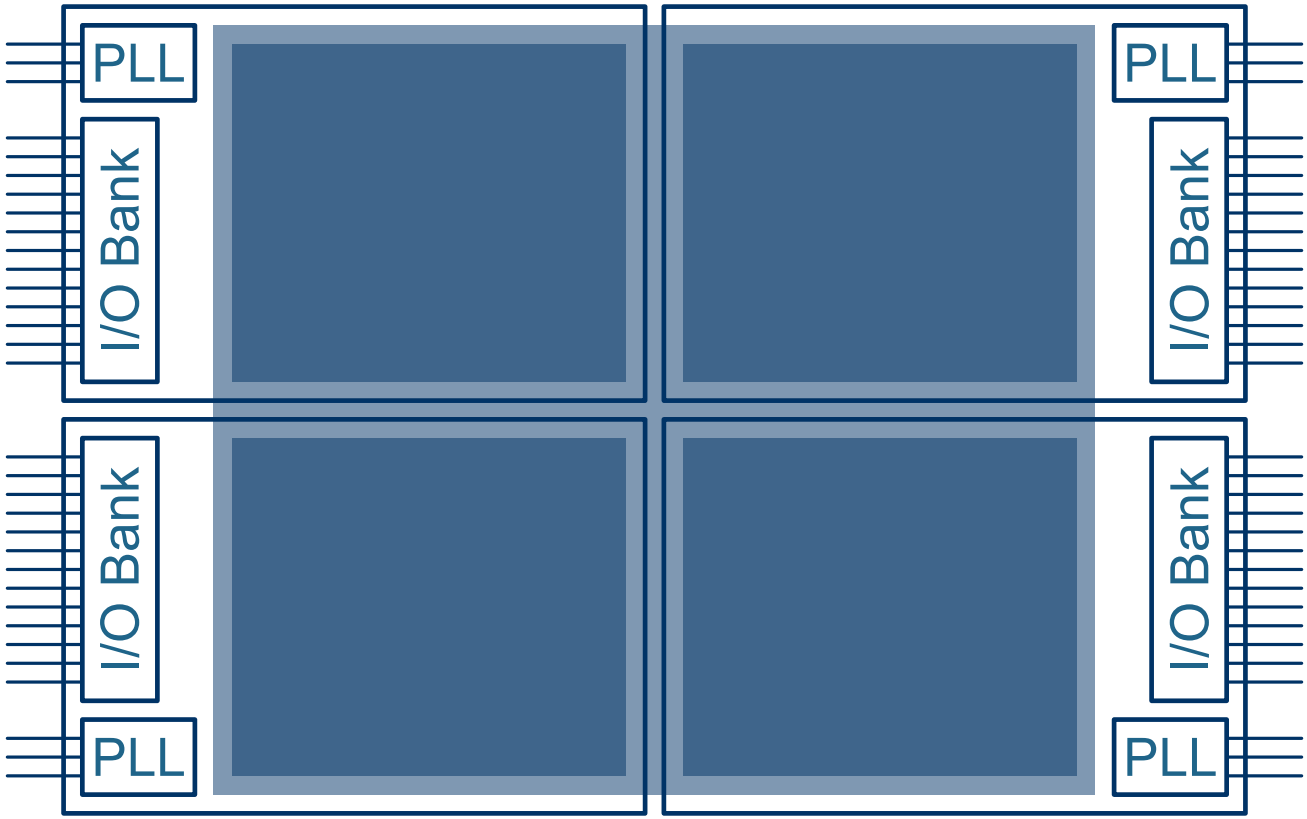
- Unrelated to all other clocks in the system, including the parent clock
- Often necessitates complex clock-domain crossing schemes to the general system clock
- Useful when:
  - Requiring very low power usage
  - The frequency must change often
  - The clock must drive an external device
  - etc.

Ripple and Gated Clocks

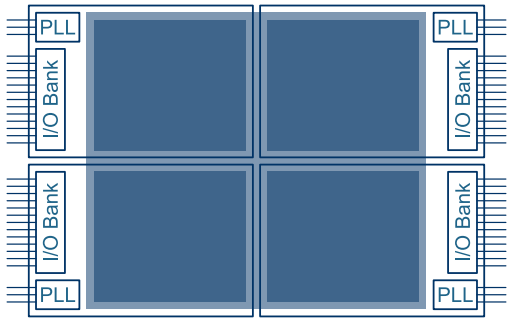


- Unrelated to all other clocks in the system, including the parent clock
- Often necessitates complex clock-domain crossing schemes to the general system clock
- Useful when:
  - Requiring very low power usage
  - The frequency must change often
  - The clock must drive an external device
  - etc.

Clock Regions

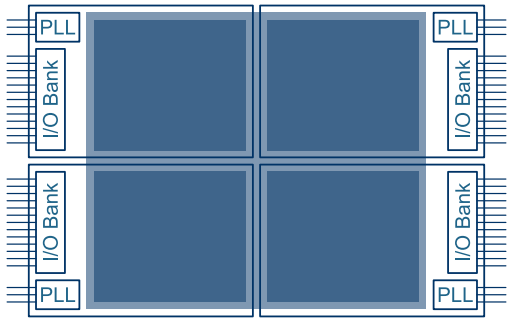


Clock Regions



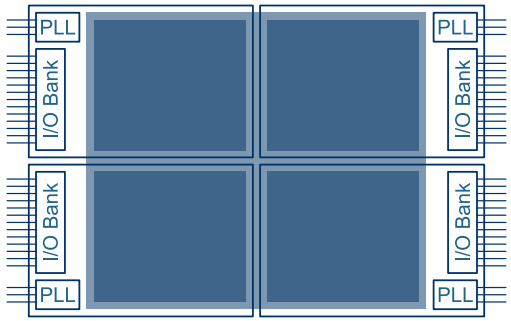
- FPGAs are generally organised into regions, each with its own PLL and I/O bank
- The local clock network is more efficient (and faster) than the global network
- When laying out the PCB, keep fast I/O in the same region as their clock

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

Advanced Clocking

Clock Domains

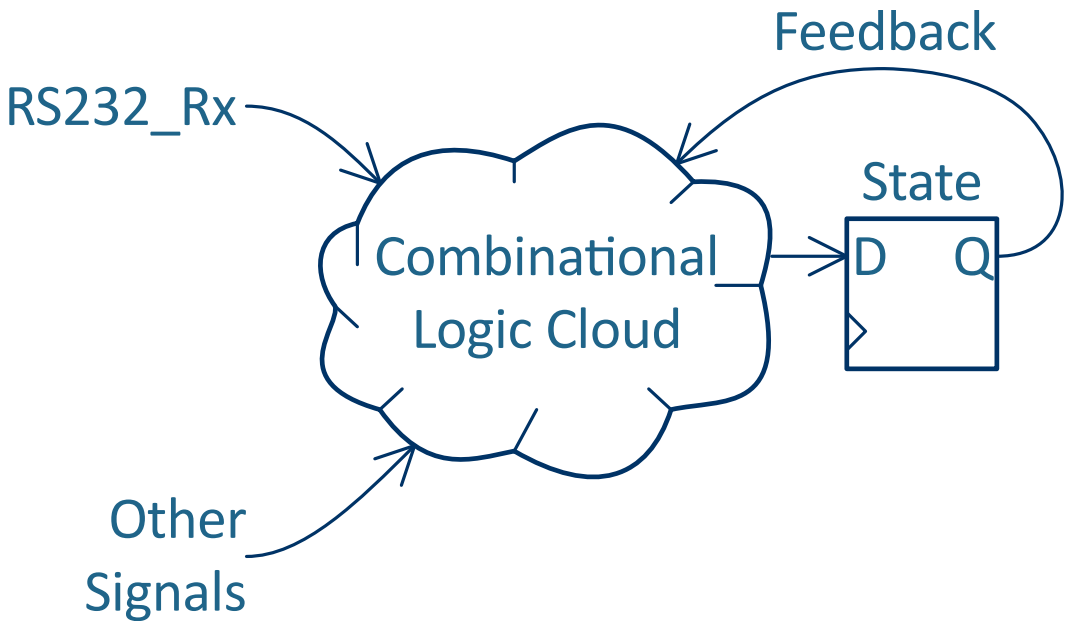
Practical

THE RADAR MASTERS COURSE

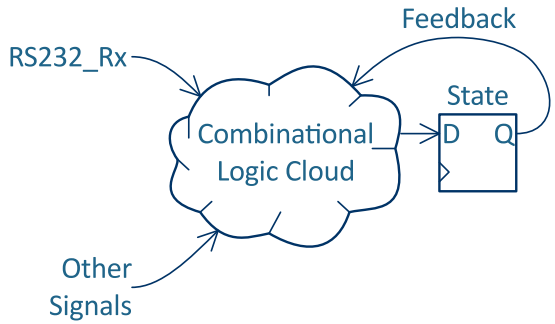
Common Mistake

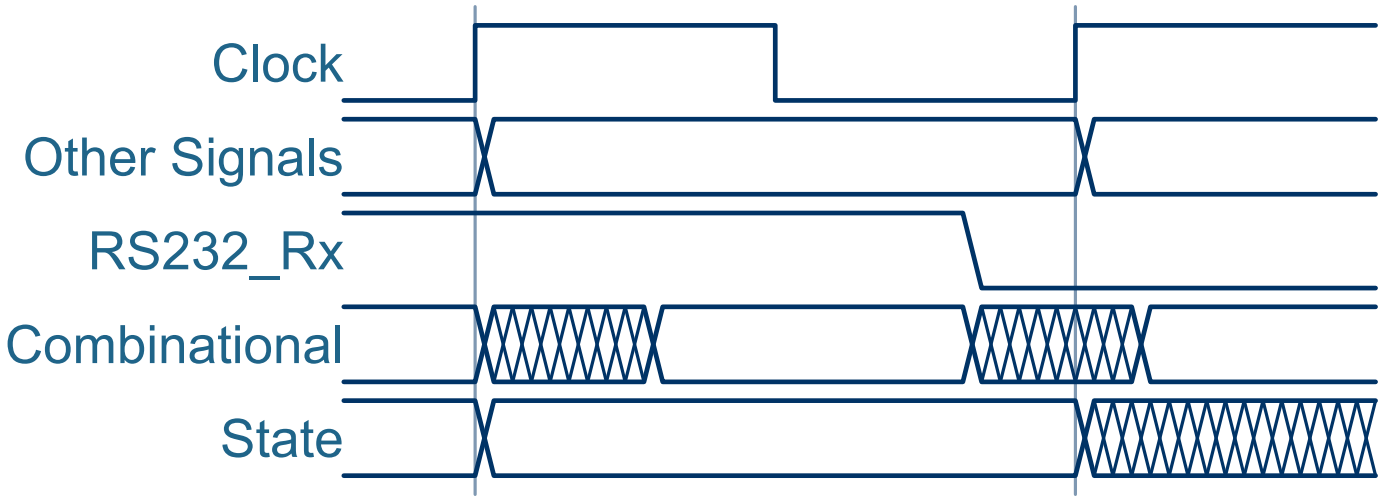

```
module RS232(
 input  Clk, Reset,
 input  RS232_Rx, // Asynchronous external signal
 output [7:0]Data
);

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

 Idle: begin
  if(!RS232_Rx) begin
   State <= Receiving;
  end
 end

 Receiving: begin
...
```

Common Mistake

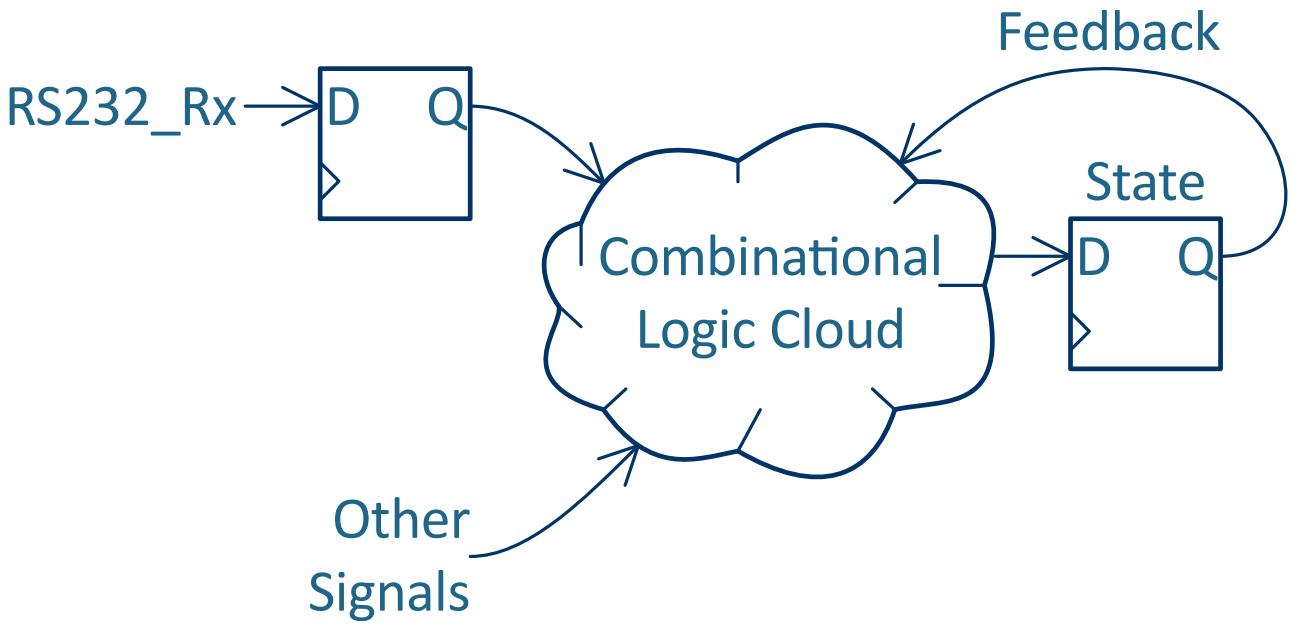


Common Mistake

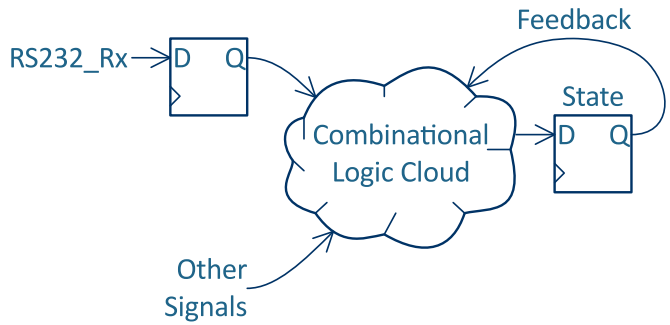


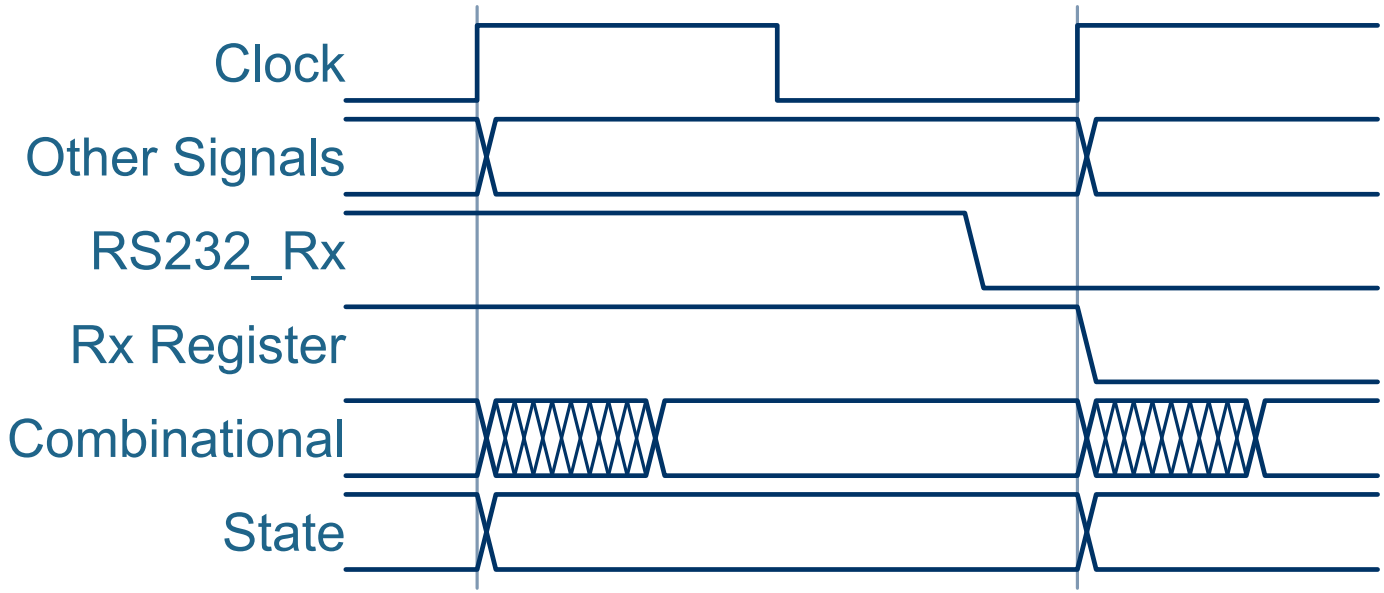


The Solution

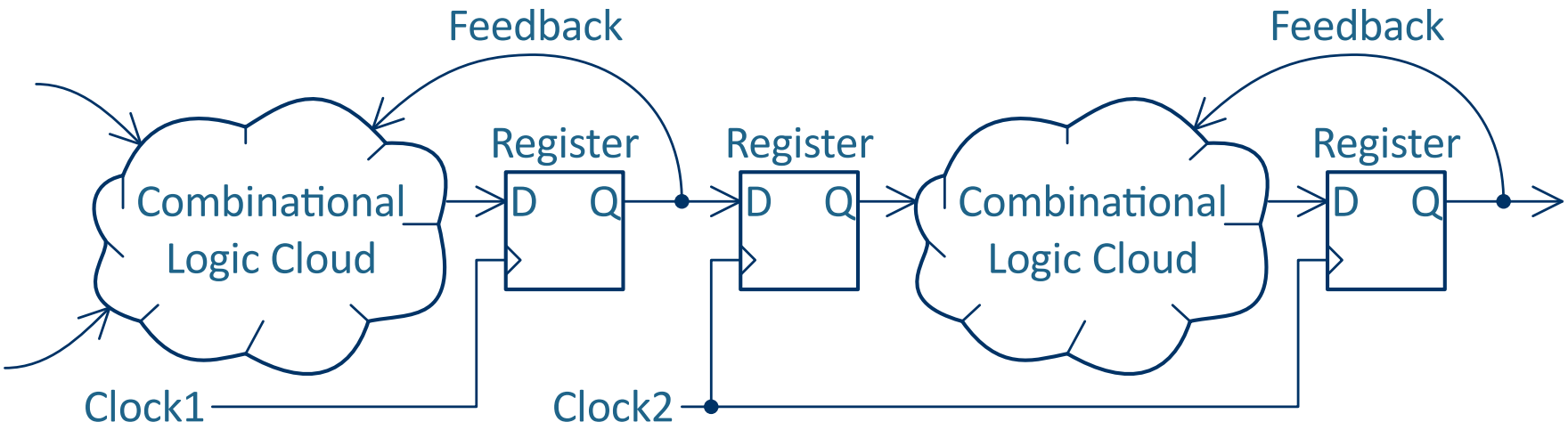


Common Mistake

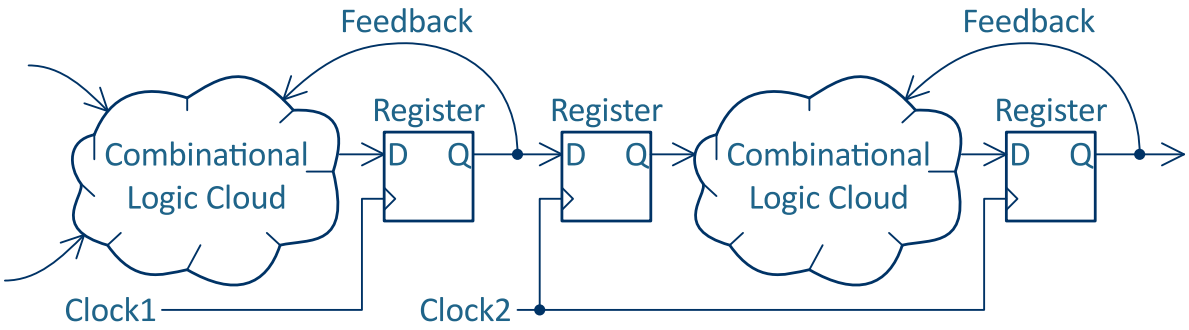




Register Chain

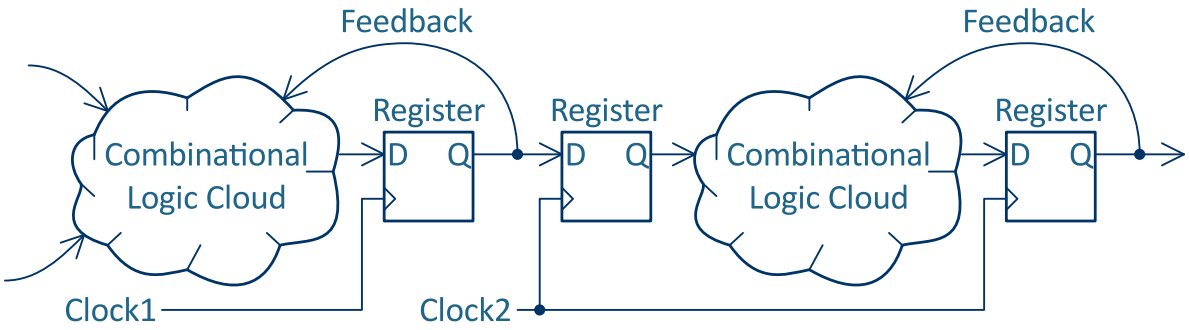


Register Chain



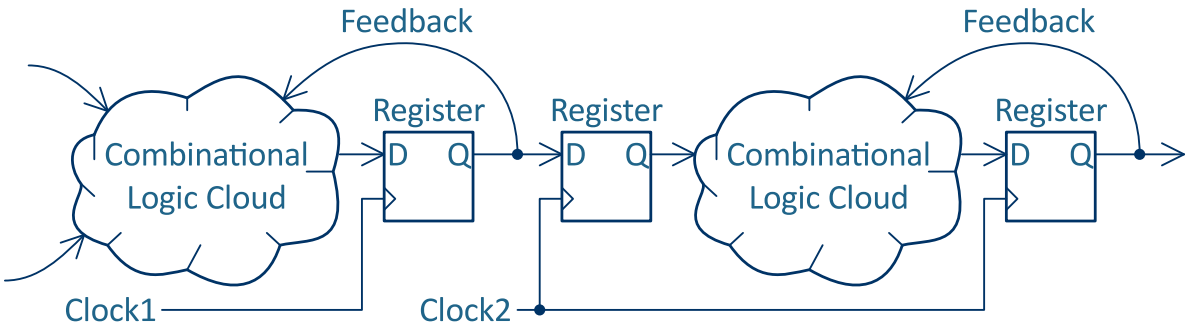
- Essential for crossing asynchronous external signals
- Only useful for crossing one bit at a time (unless the system is tolerant to temporary errors)
- Often used to cross hand-shaking signals
- Works in both directions (fast to slow / slow to fast)

Register Chain



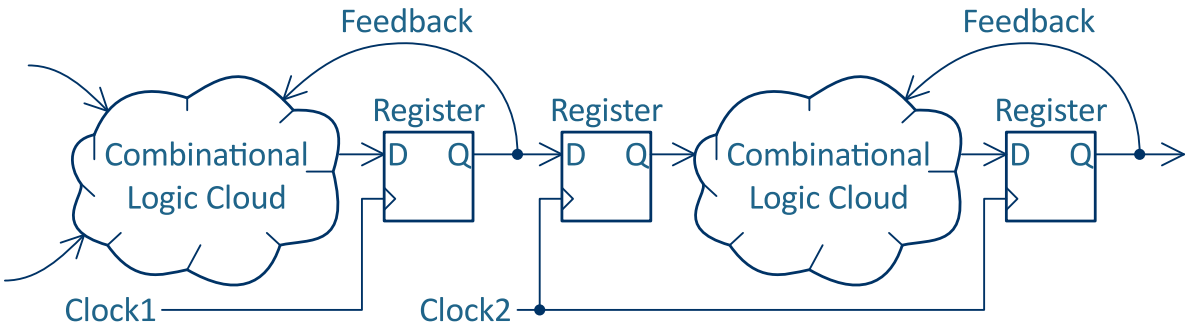
- Essential for crossing asynchronous external signals
- Only useful for crossing one bit at a time (unless the system is tolerant to temporary errors)
- Often used to cross hand-shaking signals
- Works in both directions (fast to slow / slow to fast)

Register Chain



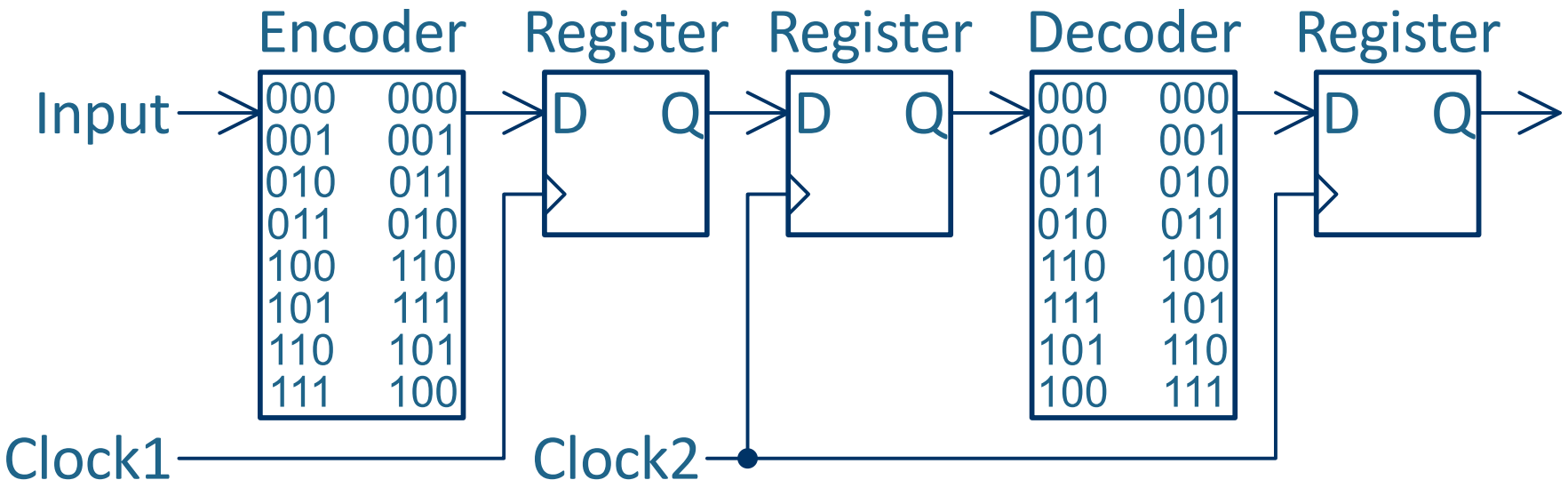
- Essential for crossing asynchronous external signals
- Only useful for crossing one bit at a time (unless the system is tolerant to temporary errors)
- Often used to cross hand-shaking signals
- Works in both directions (fast to slow / slow to fast)

Register Chain

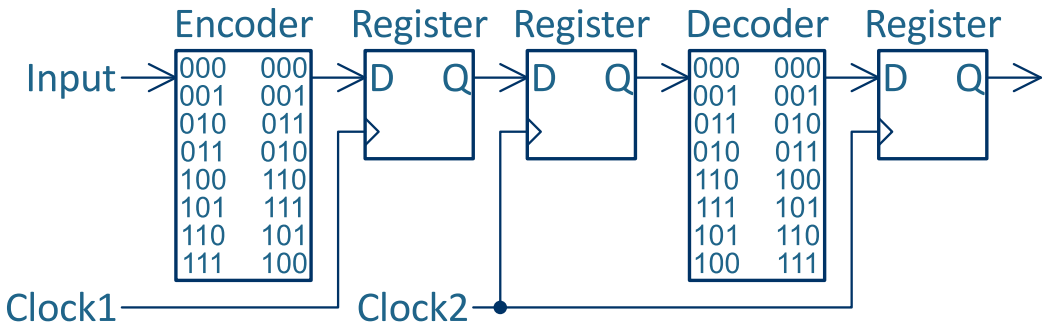


- Essential for crossing asynchronous external signals
- Only useful for crossing one bit at a time (unless the system is tolerant to temporary errors)
- Often used to cross hand-shaking signals
- Works in both directions (fast to slow / slow to fast)

Gray Coding

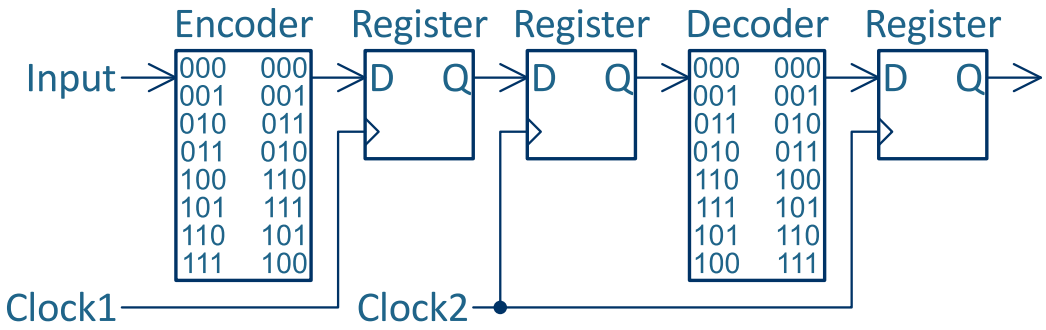

Encoder
Register
Register
Decoder
Register
Input
000 000
001 001
010 011
011 010
100 110
101 111
110 101
111 100
D Q
D Q
000 000
001 001
011 010
010 011
110 100
111 101
101 110
100 111
D Q
Clock1
Clock2

Gray Coding



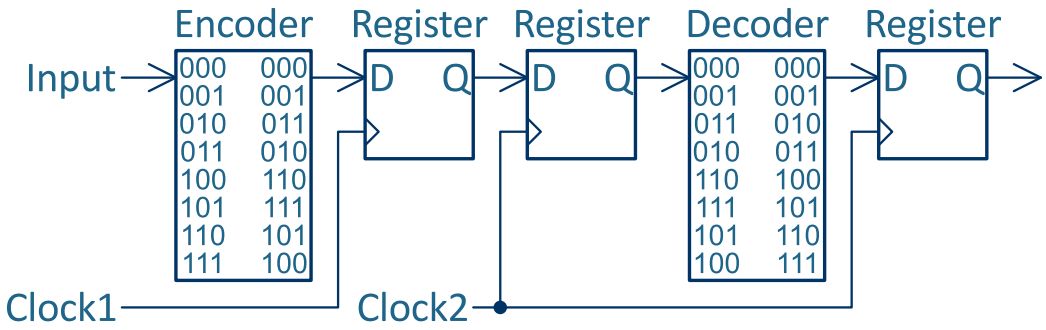
- Encoded such that only one bit changes at a time
- Only works for counter data
  (FIFO queue pointers, rotary encoders, etc.)
- Only works in one direction: slow to fast

Gray Coding



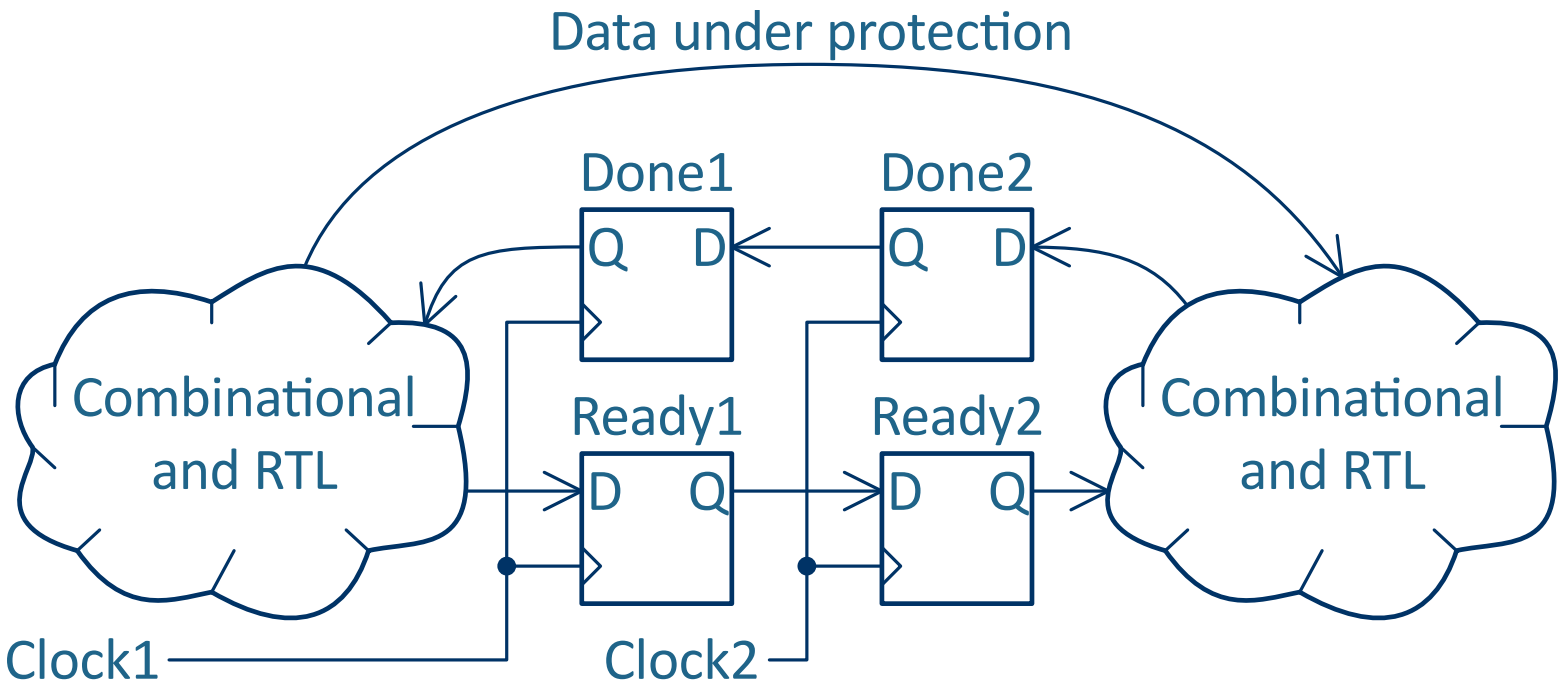
- Encoded such that only one bit changes at a time
- Only works for counter data
  (FIFO queue pointers, rotary encoders, etc.)
- Only works in one direction: slow to fast

Gray Coding

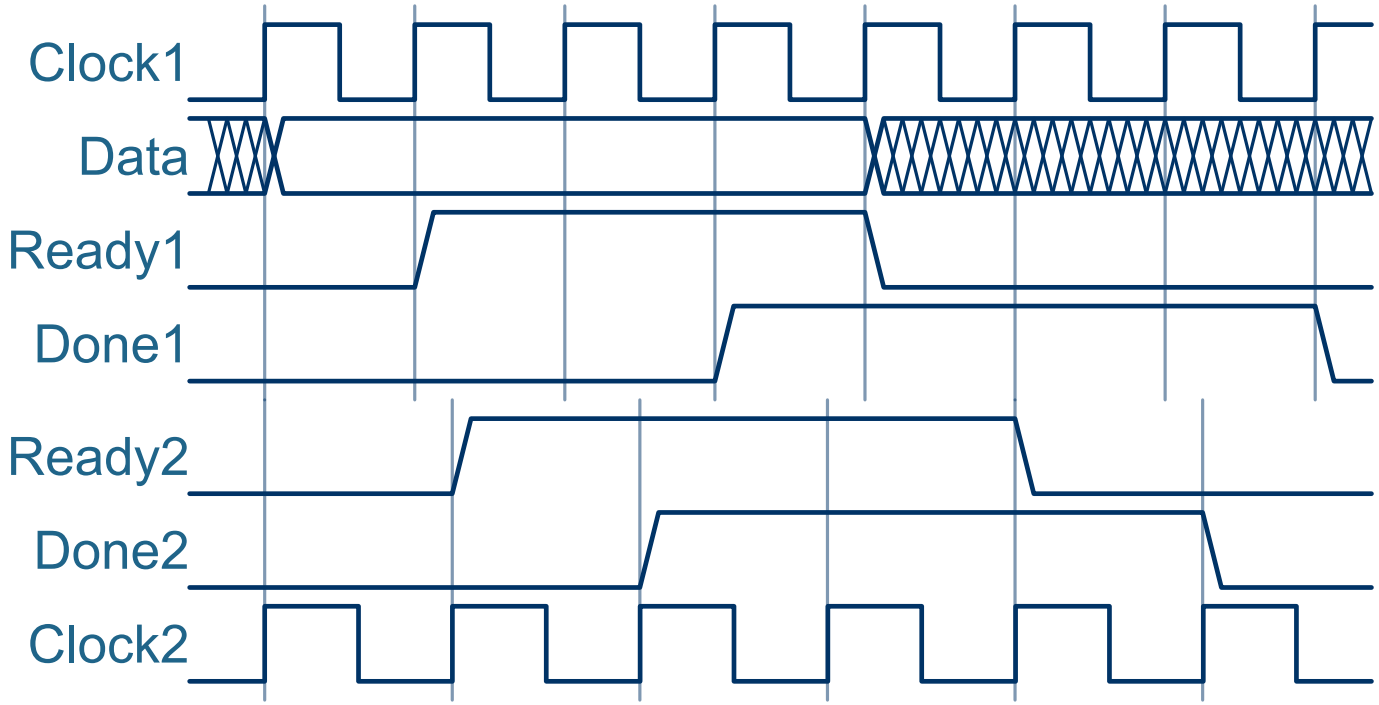


- Encoded such that only one bit changes at a time
- Only works for counter data
  (FIFO queue pointers, rotary encoders, etc.)
- Only works in one direction: slow to fast

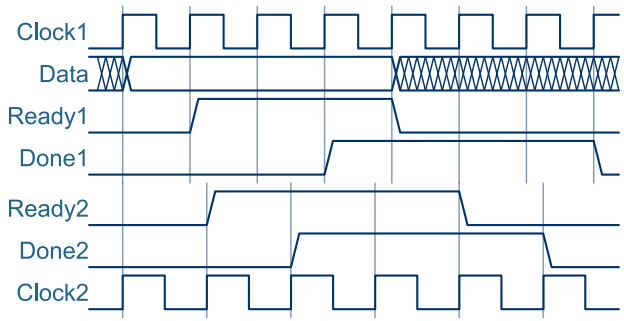
Hand-Shaking



Hand-Shaking

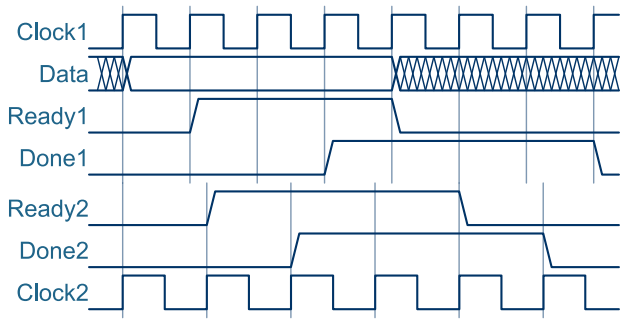


Hand-Shaking



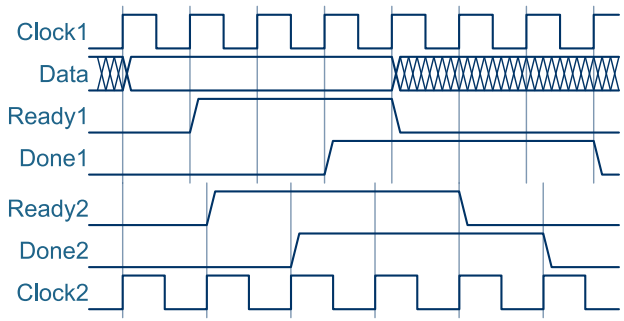
- Source must guarantee data stability while the handshaking is taking place
- Not time-efficient: need to wait for the entire transaction sequence to finish before starting the next one
- Use only for data that does not need to be crossed often
- Works in both directions (fast to slow / slow to fast)

Hand-Shaking



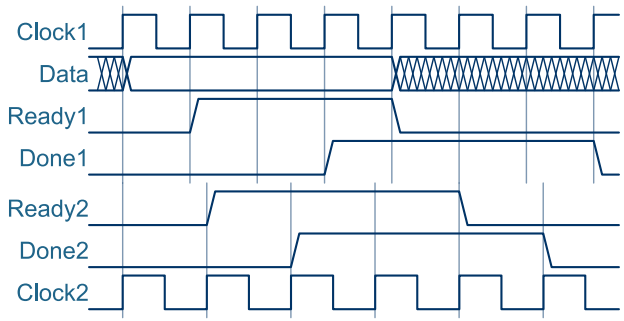
- Source must guarantee data stability while the handshaking is taking place
- Not time-efficient: need to wait for the entire transaction sequence to finish before starting the next one
- Use only for data that does not need to be crossed often
- Works in both directions (fast to slow / slow to fast)

Hand-Shaking



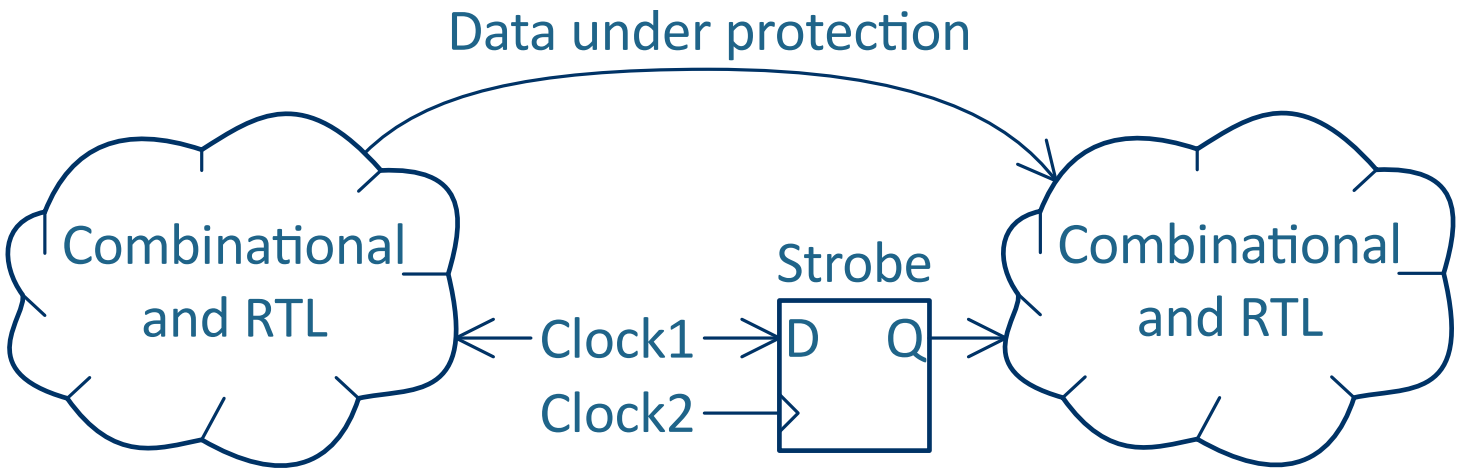
- Source must guarantee data stability while the handshaking is taking place
- Not time-efficient: need to wait for the entire transaction sequence to finish before starting the next one
- Use only for data that does not need to be crossed often
- Works in both directions (fast to slow / slow to fast)

Hand-Shaking

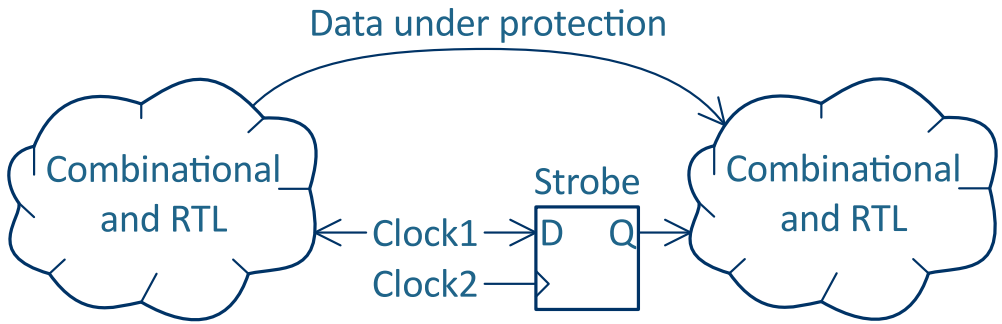


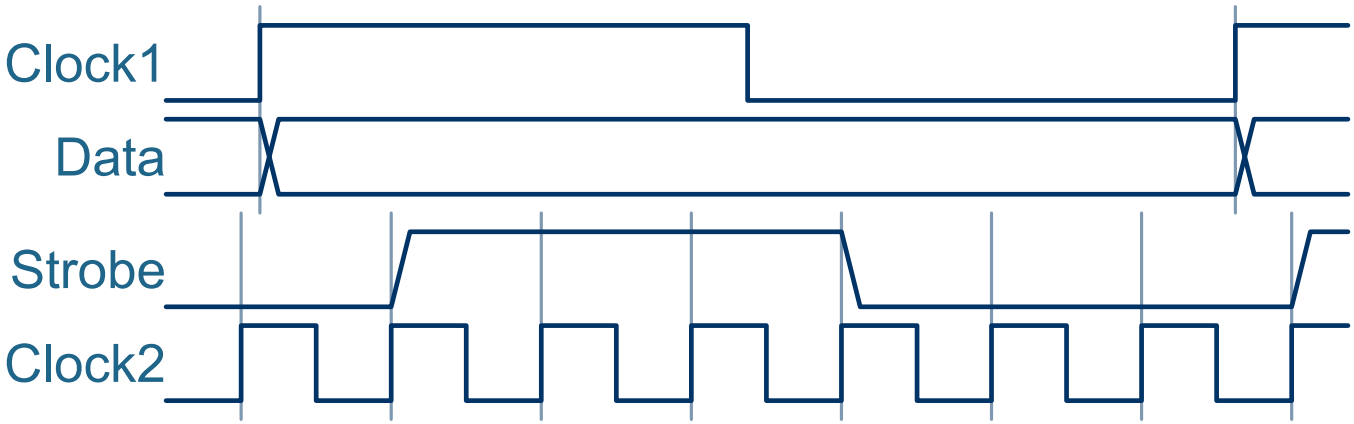
- Source must guarantee data stability while the handshaking is taking place
- Not time-efficient: need to wait for the entire transaction sequence to finish before starting the next one
- Use only for data that does not need to be crossed often
- Works in both directions (fast to slow / slow to fast)

Strobe-based

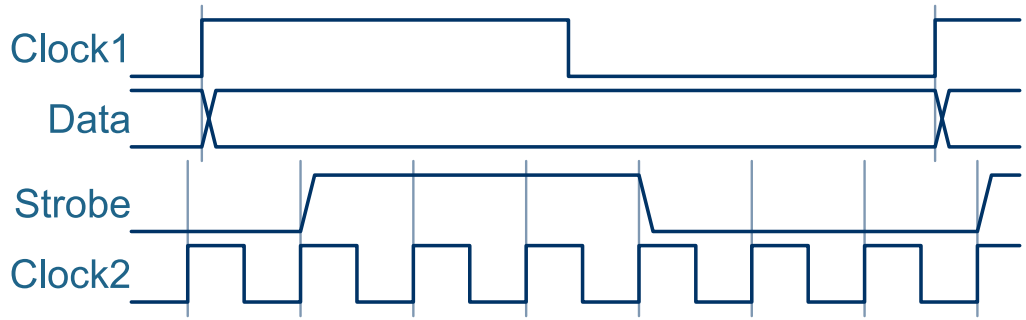


Strobe-based



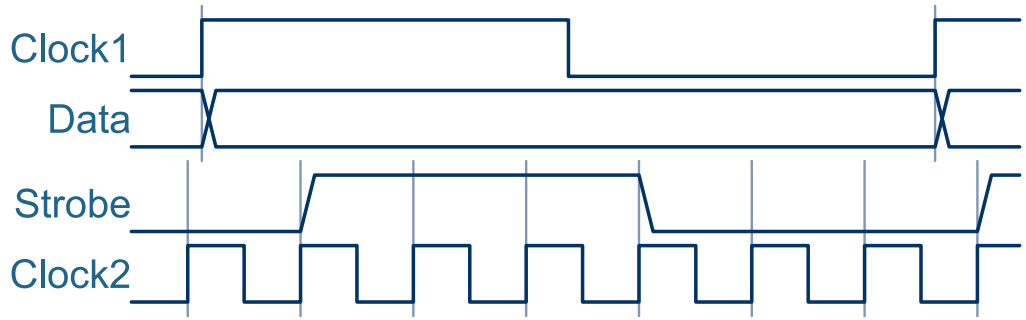


Strobe-based



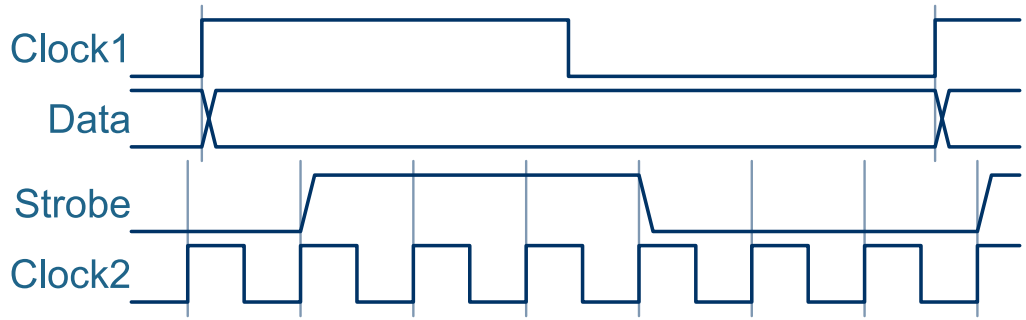
- The data is valid for as long as the strobe is high
- As fast as the source clock
- The strobe edges can be used to perform flow-control
- Only works when the destination clock is much faster than the source clock
- Note that the source “clock” can be a strobe signal generated by the source state machine...

Strobe-based



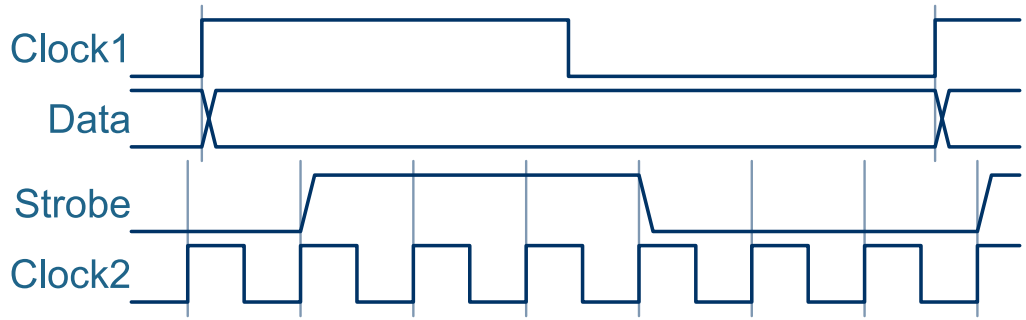
- The data is valid for as long as the strobe is high
- As fast as the source clock
- The strobe edges can be used to perform flow-control
- Only works when the destination clock is much faster than the source clock
- Note that the source “clock” can be a strobe signal generated by the source state machine...

Strobe-based



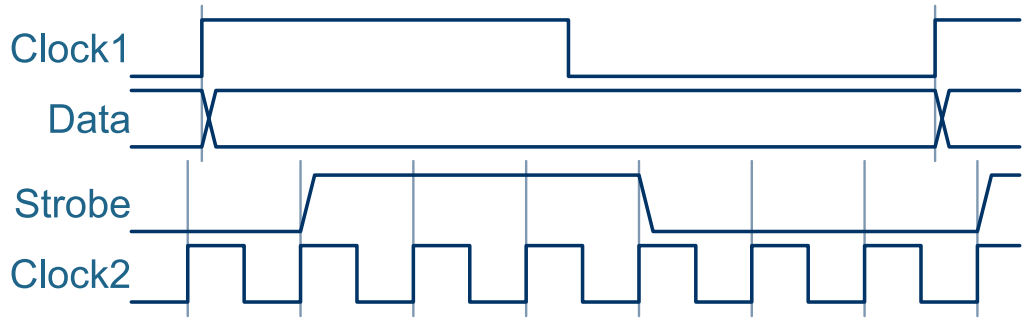
- The data is valid for as long as the strobe is high
- As fast as the source clock
- The strobe edges can be used to perform flow-control
- Only works when the destination clock is much faster than the source clock
- Note that the source “clock” can be a strobe signal generated by the source state machine...

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 -logically_exclusive \
  -group [get_clocks ADC_CLK_10]      \
  -group [get_clocks MAX10_CLK1_50]   \
  -group [get_clocks MAX10_CLK2_50]   \
  -group [get_clocks {DRAM_CLK *altpll_0*}]
```

Coffee Break...

Outline

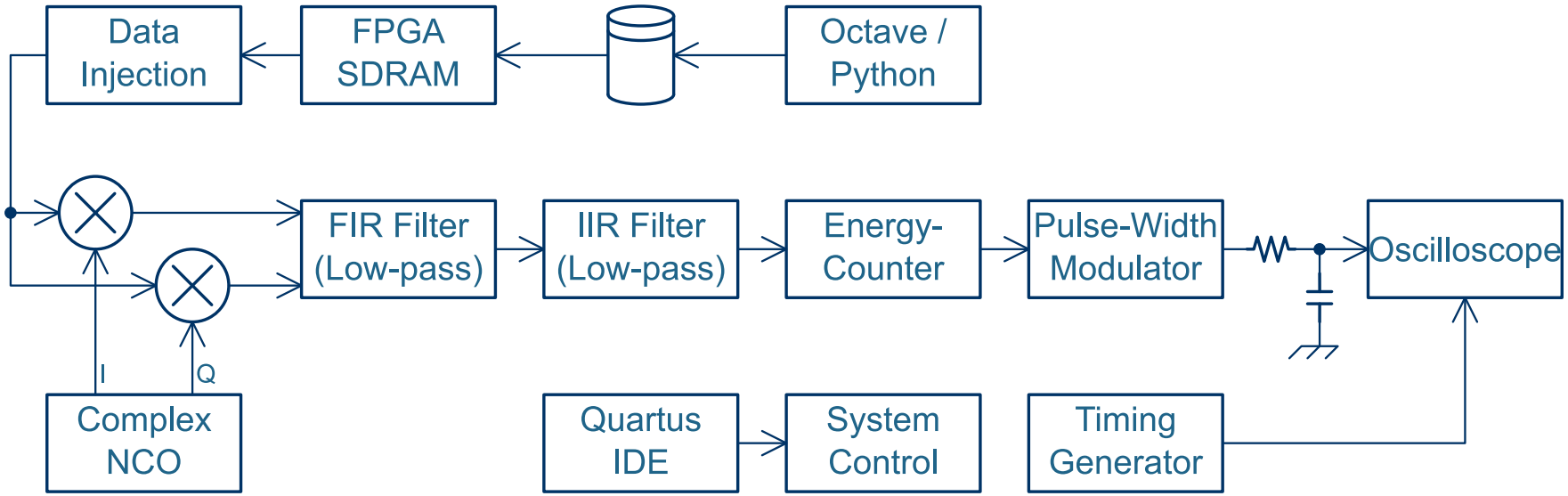
Advanced Clocking

Clock Domains

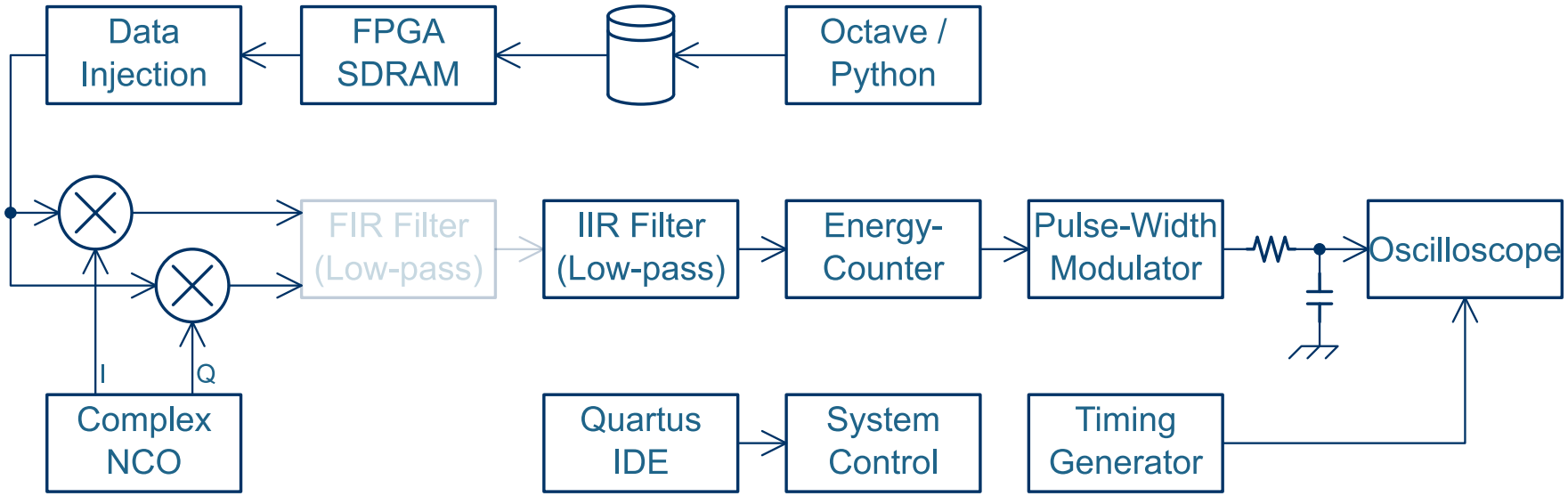
Practical

THE RADAR MASTERS COURSE

Practical


Data Injection
FPGA SDRAM
Octave / Python
FIR Filter (Low-pass)
IIR Filter (Low-pass)
Energy-Counter
Pulse-Width Modulator
Oscilloscope
I
Q
Complex NCO
Quartus IDE
System Control
Timing Generator

Practical


Data Injection
FPGA SDRAM
Octave / Python
FIR Filter (Low-pass)
IIR Filter (Low-pass)
Energy-Counter
Pulse-Width Modulator
Oscilloscope
I
Q
Complex NCO
Quartus IDE
System Control
Timing Generator

Agenda

1. Design the IIR filter in Matlab / Octave / Python (pay careful attention to potential rounding errors and the resolution required)
2. Implement the IIR filter on the 781.25 kSps clock
3. Verify by means of simulation
4. Integrate the IIR filter into the system
5. Implement the energy-counter module
6. Implement a control module to automatically sweep the NCO frequency and generate a “start-of-sweep” pulse for use as the oscilloscope trigger
7. Control the system by means of registers

Agenda

1. Design the IIR filter in Matlab / Octave / Python
2. Implement the IIR filter on the 781.25 kSps clock
3. Verify by means of simulation
4. Integrate the IIR filter into the system
5. Implement the energy-counter module
6. Implement a control module to automatically sweep the NCO frequency and generate a “start-of-sweep” pulse for use as the oscilloscope trigger
7. Control the system by means of registers

Agenda

1. Design the IIR filter in Matlab / Octave / Python
2. Implement the IIR filter on the 781.25 kSps clock
3. Verify by means of simulation (use cut-off frequencies of 10 Hz, 100 Hz, 1 kHz, 10 kHz and 100 kHz)
4. Integrate the IIR filter into the system
5. Implement the energy-counter module
6. Implement a control module to automatically sweep the NCO frequency and generate a “start-of-sweep” pulse for use as the oscilloscope trigger
7. Control the system by means of registers

Agenda

1. Design the IIR filter in Matlab / Octave / Python
2. Implement the IIR filter on the 781.25 kSps clock
3. Verify by means of simulation
4. Integrate the IIR filter into the system (simulate the FIR filter's decimation by simply re-sampling the 100 MSps data into the 781.25 kHz clock domain)
5. Implement the energy-counter module
6. Implement a control module to automatically sweep the NCO frequency and generate a "start-of-sweep" pulse for use as the oscilloscope trigger
7. Control the system by means of registers

Agenda

1. Design the IIR filter in Matlab / Octave / Python
2. Implement the IIR filter on the 781.25 kSps clock
3. Verify by means of simulation
4. Integrate the IIR filter into the system
5. Implement the energy-counter module (verify through simulation)
6. Implement a control module to automatically sweep the NCO frequency and generate a “start-of-sweep” pulse for use as the oscilloscope trigger
7. Control the system by means of registers

Agenda

1. Design the IIR filter in Matlab / Octave / Python
2. Implement the IIR filter on the 781.25 kSps clock
3. Verify by means of simulation
4. Integrate the IIR filter into the system
5. Implement the energy-counter module
6. Implement a control module to automatically sweep the NCO frequency and generate a “start-of-sweep” pulse for use as the oscilloscope trigger
7. Control the system by means of registers

Agenda

1. Design the IIR filter in Matlab / Octave / Python
2. Implement the IIR filter on the 781.25 kSps clock
3. Verify by means of simulation
4. Integrate the IIR filter into the system
5. Implement the energy-counter module
6. Implement a control module to automatically sweep the NCO frequency and generate a “start-of-sweep” pulse for use as the oscilloscope trigger
7. Control the system by means of registers (Your choice of Source-and-Probes / Virtual JTAG / etc.)

IIR Filter

$$H(s) = \frac{\omega_0^2}{s^2 + 2\zeta\omega_0 s + \omega_0^2}, \quad \omega_0 = 2\pi f_0, \quad \zeta = \frac{1}{\sqrt{2}}$$

$$s = (1 - z^{-1}) f_s, \quad f_s = 781.25\ \text{kHz}$$

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\omega_0^2}{a - bz^{-1} + cz^{-2}}, \quad \begin{cases} a = f_s^2 + 2\zeta\omega_0 f_s + \omega_0^2 \\ b = 2f_s^2 + 2\zeta\omega_0 f_s \\ c = f_s^2 \end{cases}$$

$$Y(z)\left(a - bz^{-1} + cz^{-2}\right) = X(z)\left(\omega_0^2\right)$$

$$y_n = \left(\frac{2^N}{a}\right)\left(\frac{\omega_0^2 x_n + by_{n-1} - cy_{n-2}}{2^N}\right)$$

IIR Filter

$$H(s) = \frac{\omega_0^2}{s^2 + 2\zeta\omega_0 s + \omega_0^2}, \quad \omega_0 = 2\pi f_0, \quad \zeta = \frac{1}{\sqrt{2}}$$

$$s = (1 - z^{-1}) f_s, \quad f_s = 781.25 \text{ kHz}$$

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\omega_0^2}{a - bz^{-1} + cz^{-2}}, \quad \begin{cases} a = f_s^2 + 2\zeta\omega_0 f_s + \omega_0^2 \\ b = 2f_s^2 + 2\zeta\omega_0 f_s \\ c = f_s^2 \end{cases}$$

$$Y(z)\left(a - bz^{-1} + cz^{-2}\right) = X(z)\left(\omega_0^2\right)$$

$$y_n = \left(\frac{2^N}{a}\right)\left(\frac{\omega_0^2 x_n + by_{n-1} - cy_{n-2}}{2^N}\right)$$

IIR Filter

$$H(s) = \frac{\omega_0^2}{s^2 + 2\zeta\omega_0 s + \omega_0^2}, \quad \omega_0 = 2\pi f_0, \quad \zeta = \frac{1}{\sqrt{2}}$$

$$s = (1 - z^{-1}) f_s, \quad f_s = 781.25 \text{ kHz}$$

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\omega_0^2}{a - bz^{-1} + cz^{-2}}, \quad \begin{cases} a = f_s^2 + 2\zeta\omega_0 f_s + \omega_0^2 \\ b = 2f_s^2 + 2\zeta\omega_0 f_s \\ c = f_s^2 \end{cases}$$

$$Y(z)\left(a - bz^{-1} + cz^{-2}\right) = X(z)\left(\omega_0^2\right)$$

$$y_n = \left(\frac{2^N}{a}\right)\left(\frac{\omega_0^2 x_n + by_{n-1} - cy_{n-2}}{2^N}\right)$$

IIR Filter

$$H(s) = \frac{\omega_0^2}{s^2 + 2\zeta\omega_0 s + \omega_0^2}, \quad \omega_0 = 2\pi f_0, \quad \zeta = \frac{1}{\sqrt{2}}$$

$$s = (1 - z^{-1}) f_s, \quad f_s = 781.25 \text{ kHz}$$

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\omega_0^2}{a - bz^{-1} + cz^{-2}}, \quad \begin{cases} a = f_s^2 + 2\zeta\omega_0 f_s + \omega_0^2 \\ b = 2f_s^2 + 2\zeta\omega_0 f_s \\ c = f_s^2 \end{cases}$$

$$Y(z)\left(a - bz^{-1} + cz^{-2}\right) = X(z)\left(\omega_0^2\right)$$

$$y_n = \left(\frac{2^N}{a}\right)\left(\frac{\omega_0^2 x_n + by_{n-1} - cy_{n-2}}{2^N}\right)$$

IIR Filter

$$H(s) = \frac{\omega_0^2}{s^2 + 2\zeta\omega_0 s + \omega_0^2}, \quad \omega_0 = 2\pi f_0, \quad \zeta = \frac{1}{\sqrt{2}}$$

$$s = (1 - z^{-1}) f_s, \quad f_s = 781.25 \text{ kHz}$$

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\omega_0^2}{a - bz^{-1} + cz^{-2}}, \quad \begin{cases} a = f_s^2 + 2\zeta\omega_0 f_s + \omega_0^2 \\ b = 2f_s^2 + 2\zeta\omega_0 f_s \\ c = f_s^2 \end{cases}$$

$$Y(z)\left(a - bz^{-1} + cz^{-2}\right) = X(z)\left(\omega_0^2\right)$$

$$y_n = \left(\frac{2^N}{a}\right)\left(\frac{\omega_0^2 x_n + by_{n-1} - cy_{n-2}}{2^N}\right)$$

IIR Filter

$$y_n = \frac{Ax_n + By_{n-1} - Cy_{n-2}}{2^N}, \quad \begin{cases} A = (2^N/a) \cdot \omega_0^2 \\ B = (2^N/a) \cdot b \\ C = (2^N/a) \cdot c \\ N = 32 \end{cases}$$

f_0	A	B	C
10	28	8 589 446 092	4 294 478 824
100	2 775	8 585 049 594	4 290 085 073
1 000	274 663	8 541 087 701	4 246 395 068
10 000	24 799 428	8 104 255 079	3 834 087 211
100 000	997 792 625	4 839 800 553	1 542 625 882

IIR Filter

- To get the IIR filter to work at low pass-band frequencies, you need LARGE internal word-lengths
- Be careful with overflows – perform a limit operation
- Do the multiplication manually
  (absolute → multiply unsigned → sign return)
- To help you keep track of what bit represents what in fixed-point representations, use negative indices:

```
// Constants are in the range [0, 2), but
// Verilog still thinks they're unsigned integers
wire [0:-32]B = 33'd_8_589_446_092;
// Internal registers are in the range [-1, 1)
reg  [0:-39]y_1;
// Products are in the range [-2, 2)
wire [1:-71]B_y_1;
```

IIR Filter

- To get the IIR filter to work at low pass-band frequencies, you need LARGE internal word-lengths
- Be careful with overflows – perform a limit operation
- Do the multiplication manually
  (absolute → multiply unsigned → sign return)
- To help you keep track of what bit represents what in fixed-point representations, use negative indices:

```
// Constants are in the range [0, 2), but
// Verilog still thinks they're unsigned integers
wire [0:-32]B = 33'd_8_589_446_092;
// Internal registers are in the range [-1, 1)
reg  [0:-39]y_1;
// Products are in the range [-2, 2)
wire [1:-71]B_y_1;
```

IIR Filter

- To get the IIR filter to work at low pass-band frequencies, you need LARGE internal word-lengths
- Be careful with overflows – perform a limit operation
- Do the multiplication manually
  (absolute $\rightarrow$ multiply unsigned $\rightarrow$ sign return)
- To help you keep track of what bit represents what in fixed-point representations, use negative indices:

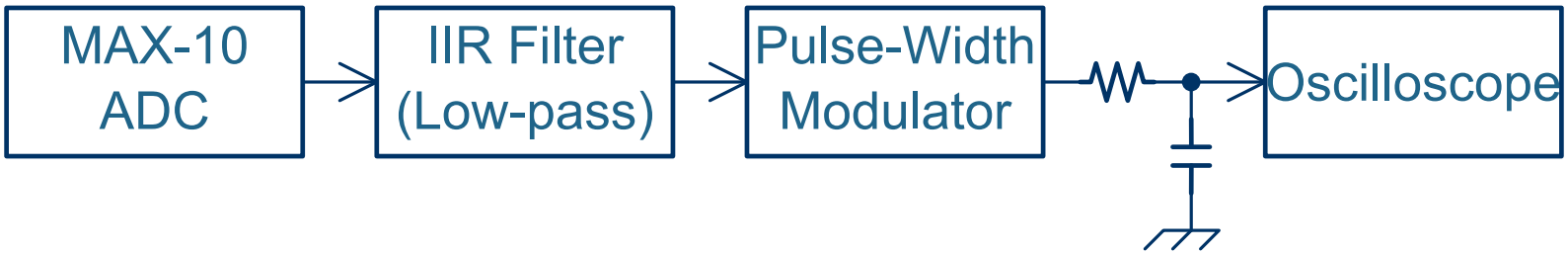
```
// Constants are in the range [0, 2), but
// Verilog still thinks they're unsigned integers
wire [0:-32]B = 33'd_8_589_446_092;
// Internal registers are in the range [-1, 1)
reg  [0:-39]y_1;
// Products are in the range [-2, 2)
wire [1:-71]B_y_1;
```

IIR Filter

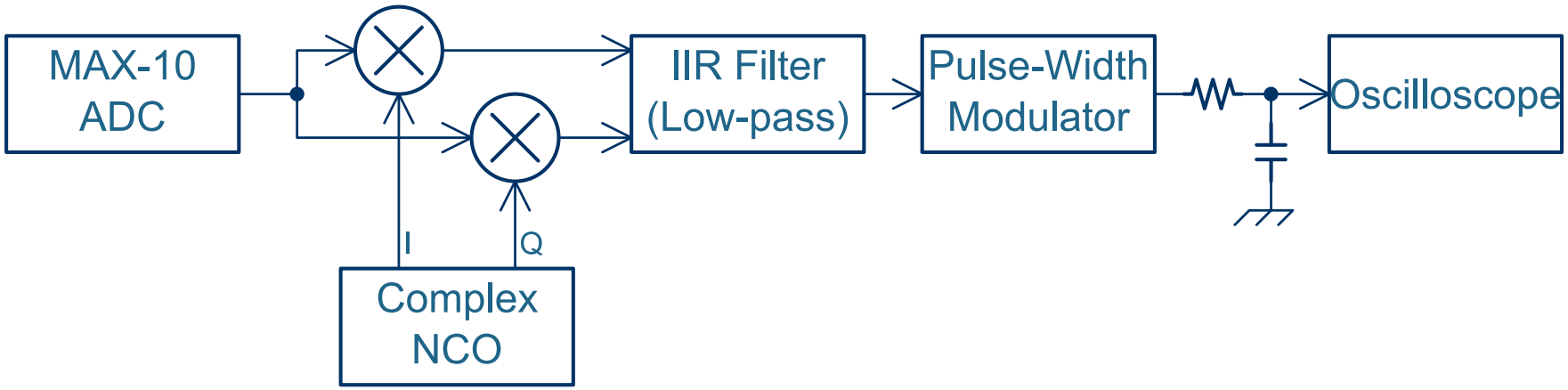
- To get the IIR filter to work at low pass-band frequencies, you need LARGE internal word-lengths
- Be careful with overflows – perform a limit operation
- Do the multiplication manually
  (absolute → multiply unsigned → sign return)
- To help you keep track of what bit represents what in fixed-point representations, use negative indices:

```
// Constants are in the range [0, 2), but
// Verilog still thinks they're unsigned integers
wire [0:-32]B = 33'd_8_589_446_092;
// Internal registers are in the range [-1, 1)
reg  [0:-39]y_1;
// Products are in the range [-2, 2)
wire [1:-71]B_y_1;
```

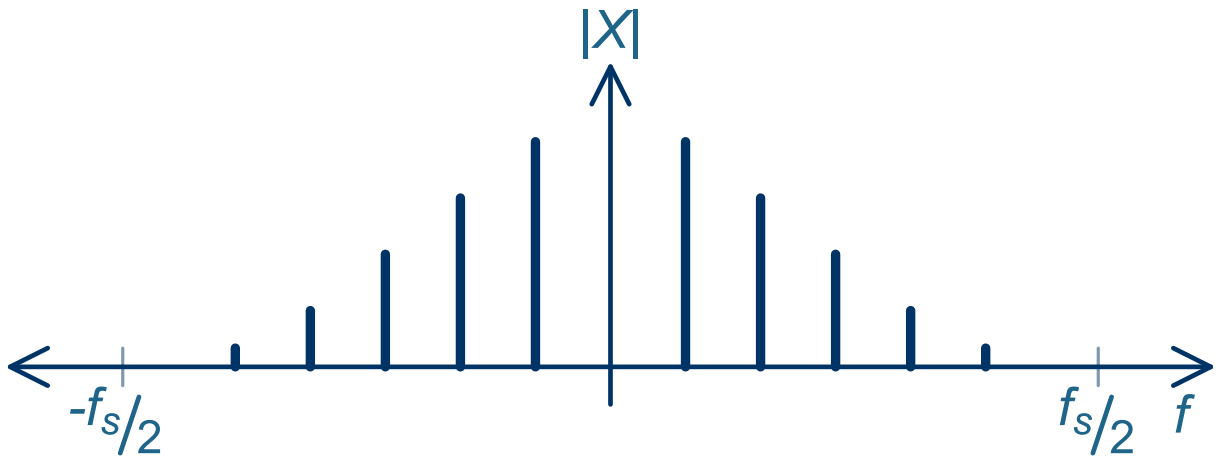
IIR Filter – ADC Test (Optional)



IIR Filter – ADC Test (Optional)



IIR Filter – Injection Test

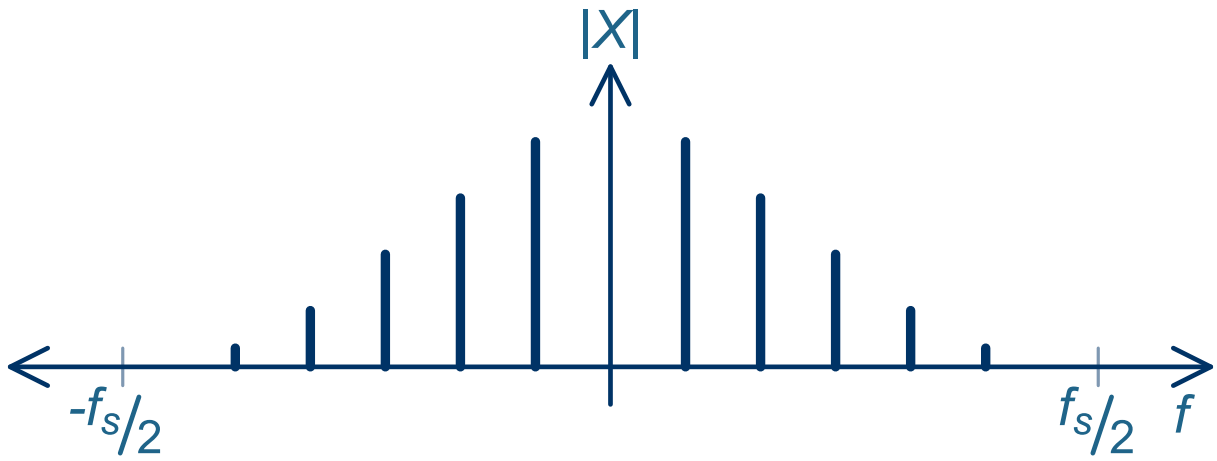


$f_s = 781.25$ kHz

$x = 60$ kHz square wave (injected or NCO)

- Make the resolution bandwidth 2 kHz (1 kHz low-pass filter)
- Sweep the complex NCO from DC to 360 kHz

IIR Filter – Injection Test

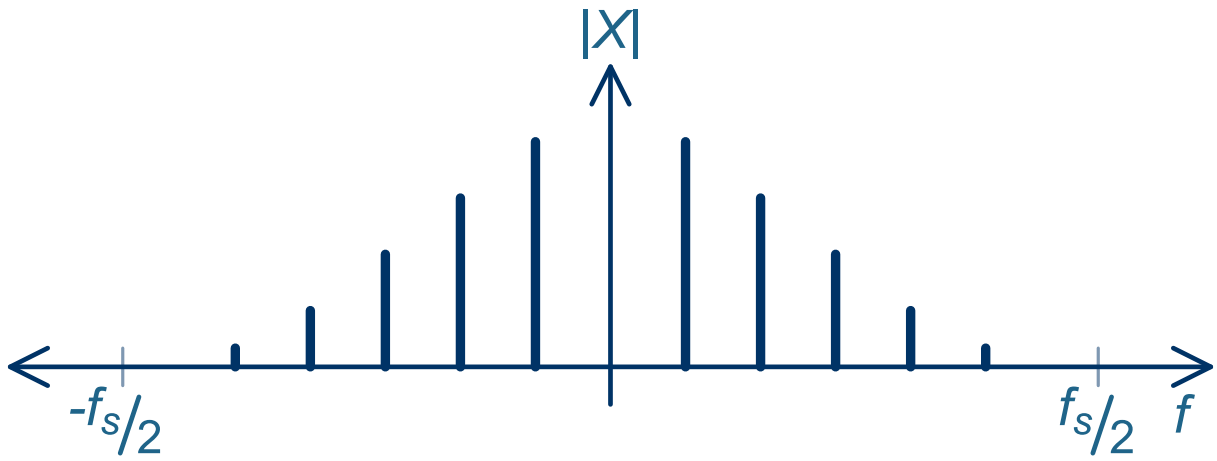


$f_s = 781.25$ kHz

$x = 60$ kHz square wave (injected or NCO)

- Make the resolution bandwidth 2 kHz (1 kHz low-pass filter)
- Sweep the complex NCO from DC to 360 kHz

IIR Filter – Injection Test

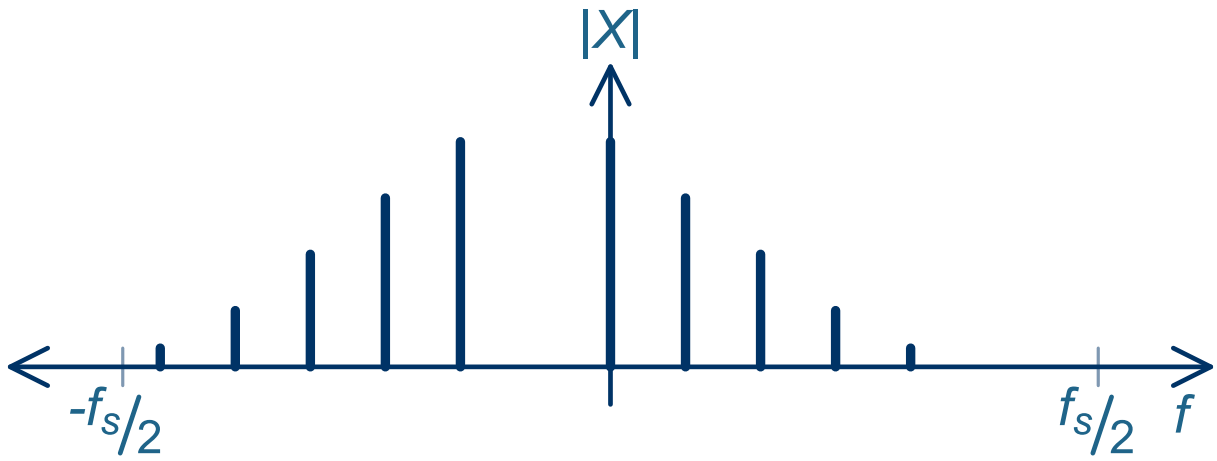


$f_s = 781.25$ kHz

$x = 60$ kHz square wave (injected or NCO)

- Make the resolution bandwidth 2 kHz (1 kHz low-pass filter)
- Sweep the complex NCO from DC to 360 kHz

IIR Filter – Injection Test

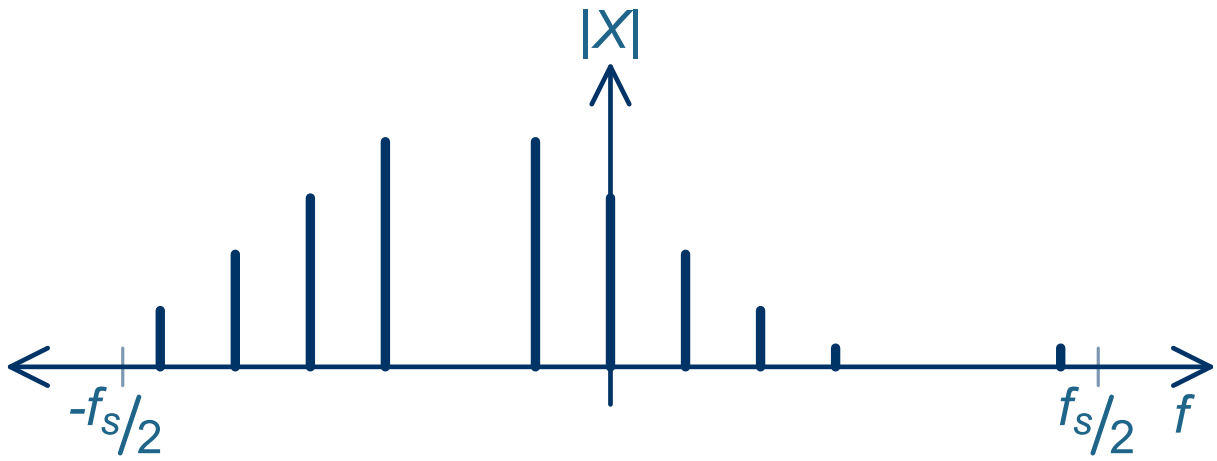


$f_s = 781.25$ kHz

$x = 60$ kHz square wave (injected or NCO)

- Make the resolution bandwidth 2 kHz (1 kHz low-pass filter)
- Sweep the complex NCO from DC to 360 kHz

IIR Filter – Injection Test

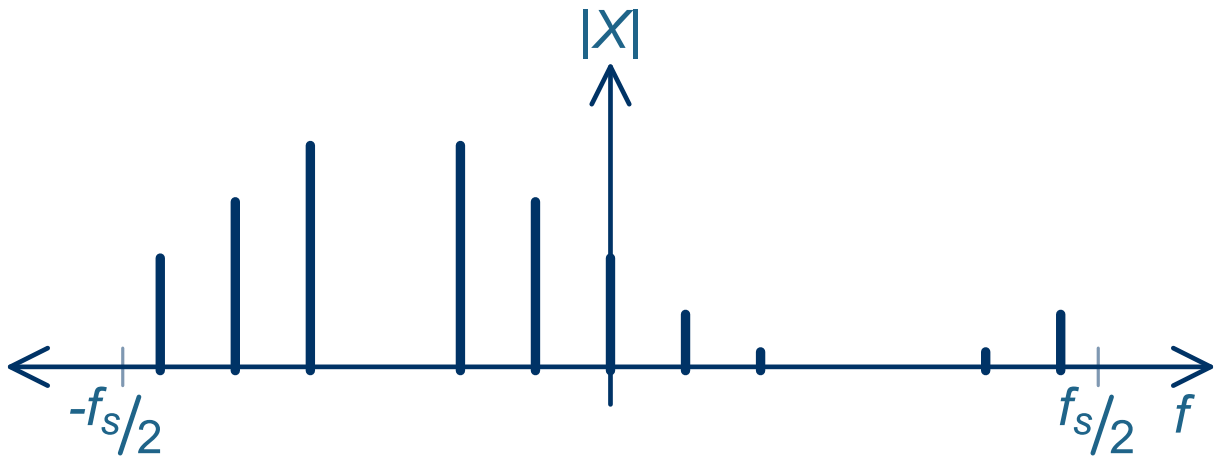


$f_s = 781.25$ kHz

$x = 60$ kHz square wave (injected or NCO)

- Make the resolution bandwidth 2 kHz (1 kHz low-pass filter)
- Sweep the complex NCO from DC to 360 kHz

IIR Filter – Injection Test

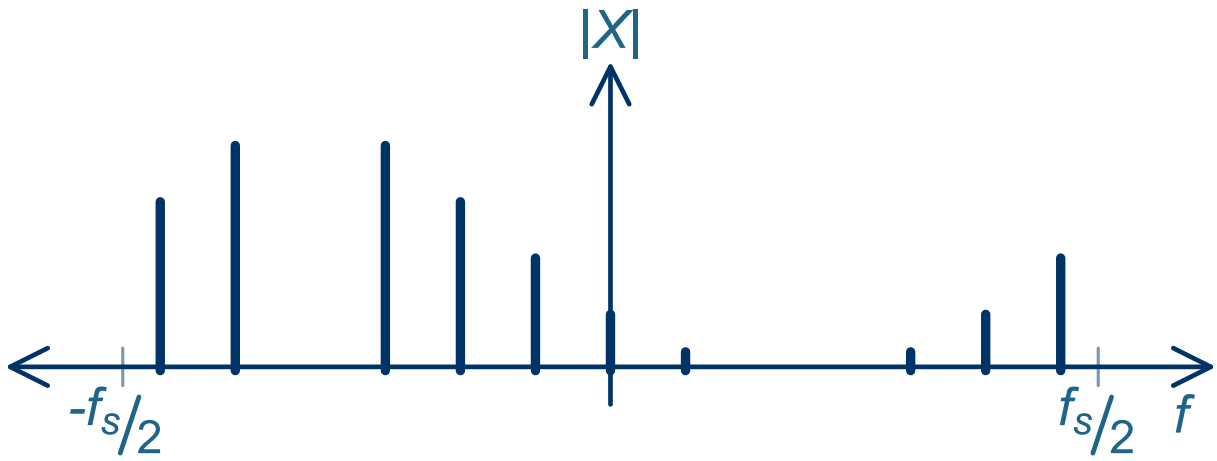


$f_s = 781.25$ kHz

$x = 60$ kHz square wave (injected or NCO)

- Make the resolution bandwidth 2 kHz (1 kHz low-pass filter)
- Sweep the complex NCO from DC to 360 kHz

IIR Filter – Injection Test

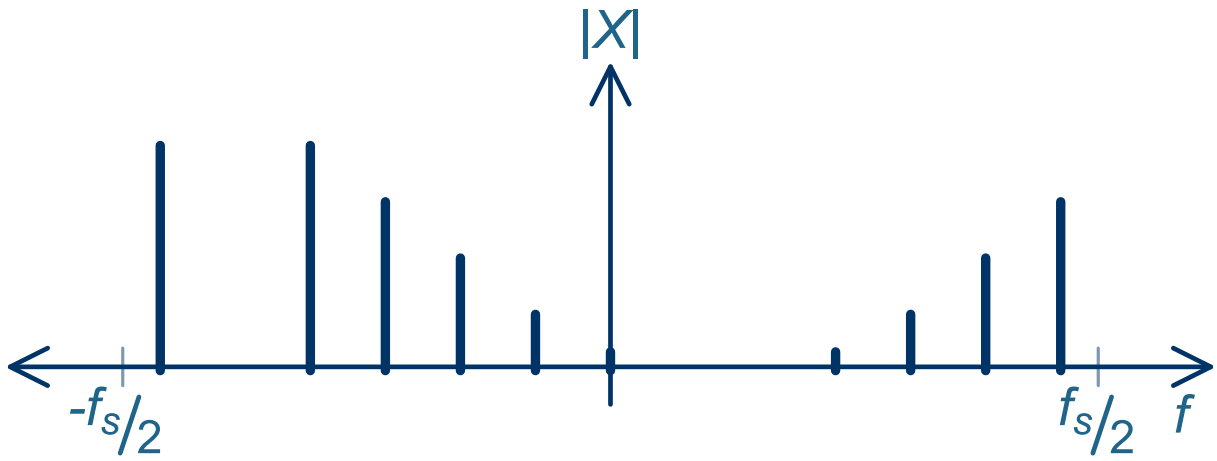


$f_s = 781.25$ kHz

$x = 60$ kHz square wave (injected or NCO)

- Make the resolution bandwidth 2 kHz (1 kHz low-pass filter)
- Sweep the complex NCO from DC to 360 kHz

IIR Filter – Injection Test

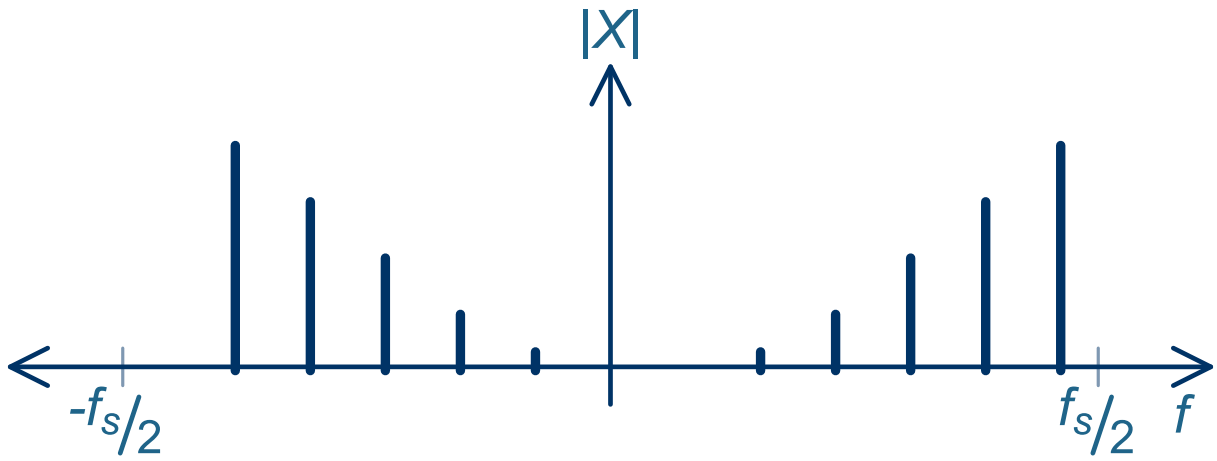


$f_s = 781.25$ kHz

$x = 60$ kHz square wave (injected or NCO)

- Make the resolution bandwidth 2 kHz (1 kHz low-pass filter)
- Sweep the complex NCO from DC to 360 kHz

IIR Filter – Injection Test



$f_s = 781.25$ kHz

$x = 60$ kHz square wave (injected or NCO)

- Make the resolution bandwidth 2 kHz (1 kHz low-pass filter)
- Sweep the complex NCO from DC to 360 kHz

Energy Counter

$$
\begin{aligned}
P &= \frac{1}{T_p} \int_{T_p} |x(t)|^2 \, \mathrm{d}t \\
&= \frac{1}{N} \sum_{n=0}^{N-1} (x_n x_n^*)
\end{aligned}
$$

- Use a variable N (time-window) – use power-of-two sizes
- Use a LUT to convert the signal power to log-scale

Energy Counter

$$
\begin{aligned}
P &= \frac{1}{T_p} \int_{T_p} |x(t)|^2 \, \mathrm{d}t \\
&= \frac{1}{N} \sum_{n=0}^{N-1} (x_n x_n^*)
\end{aligned}
$$

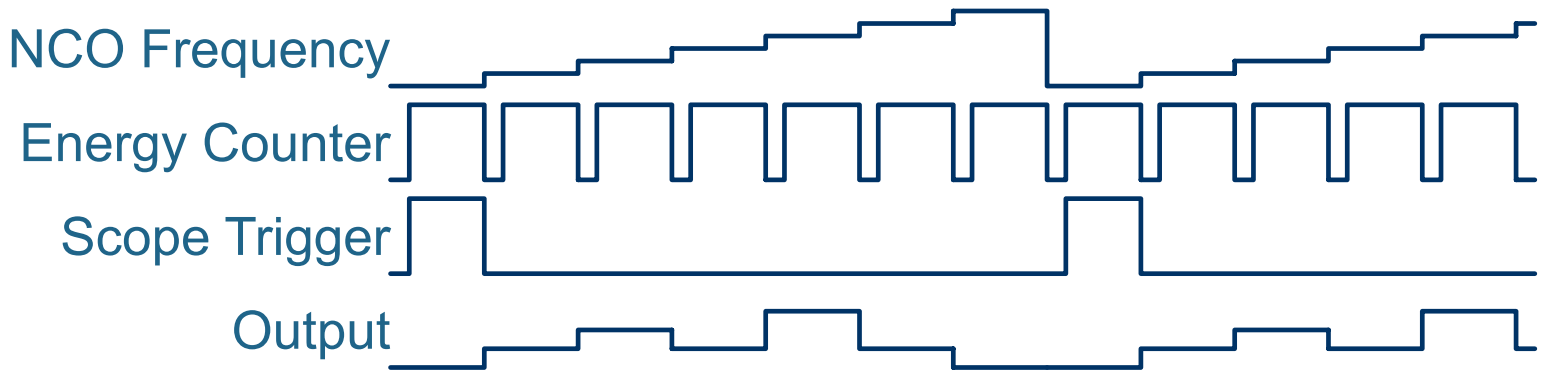
- Use a variable N (time-window) – use power-of-two sizes
- Use a LUT to convert the signal power to log-scale

Energy Counter

$$
\begin{aligned}
P &= \frac{1}{T_p} \int_{T_p} |x(t)|^2 \, \mathrm{d}t \\
&= \frac{1}{N} \sum_{n=0}^{N-1} (x_n x_n^*)
\end{aligned}
$$

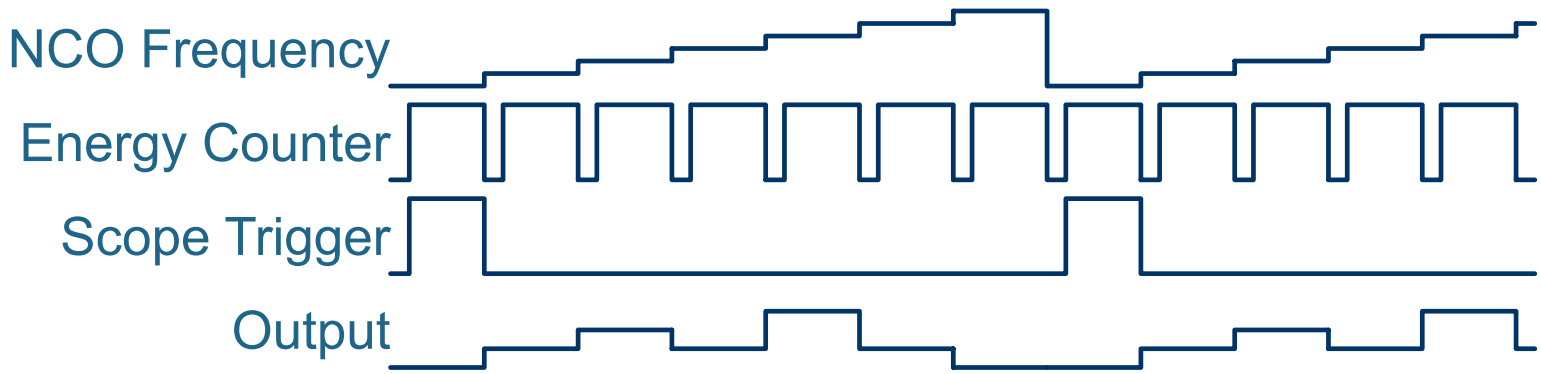
- Use a variable N (time-window) – use power-of-two sizes
- Use a LUT to convert the signal power to log-scale

Timing Generator



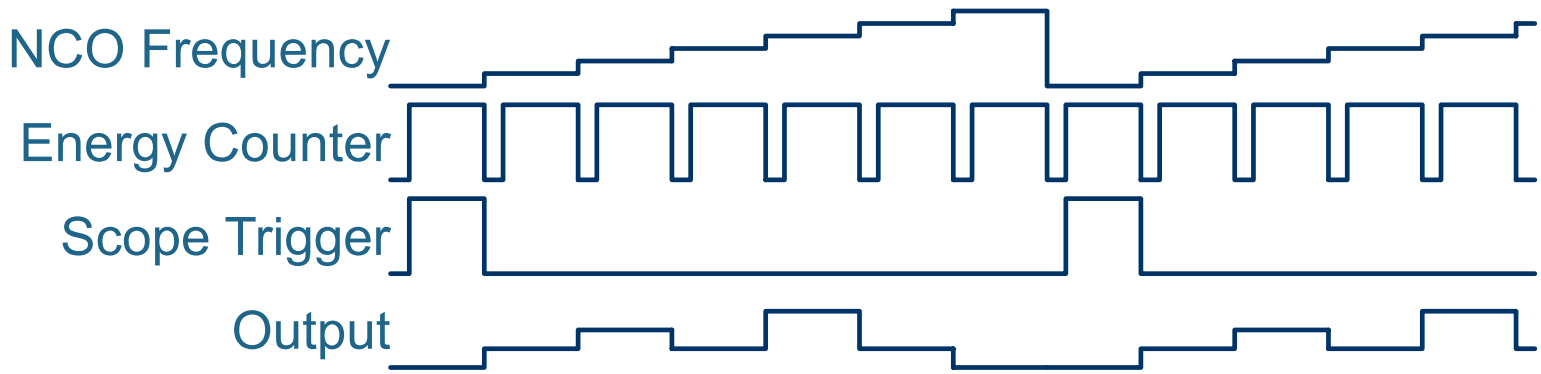
- Many parameters are involved...
- Set the parameters by means of registers: let the PC do all the calculations and sanity-checks
- Or you can use look-up tables to simplify the registers (eg. resolution bandwidth selection by constants $1 \rightarrow 5$, translated to the filter constants, energy-counter window size, counter limits, NCO step-sizes, etc.)

Timing Generator



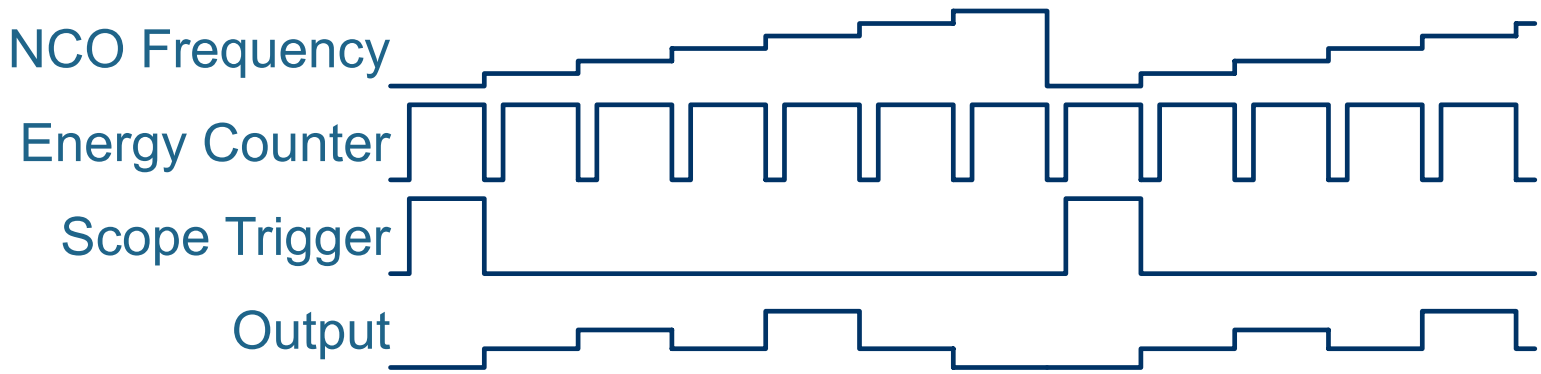
- Many parameters are involved...
- Set the parameters by means of registers: let the PC do all the calculations and sanity-checks
- Or you can use look-up tables to simplify the registers (eg. resolution bandwidth selection by constants 1 $\rightarrow$ 5, translated to the filter constants, energy-counter window size, counter limits, NCO step-sizes, etc.)

Timing Generator



- Many parameters are involved...
- Set the parameters by means of registers: let the PC do all the calculations and sanity-checks
- Or you can use look-up tables to simplify the registers (eg. resolution bandwidth selection by constants $1 \rightarrow 5$, translated to the filter constants, energy-counter window size, counter limits, NCO step-sizes, etc.)

Timing Generator



- Many parameters are involved...
- Set the parameters by means of registers: let the PC do all the calculations and sanity-checks
- Or you can use look-up tables to simplify the registers (eg. resolution bandwidth selection by constants $1 \rightarrow 5$, translated to the filter constants, energy-counter window size, counter limits, NCO step-sizes, etc.)

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 Prof Daniel O'Hagan

Tutored by Stephen Paine and Randy Cheng

Day 4 – 20 July 2017