6.S965
Digital Systems Laboratory II

Lecture 4:

Signal Processing I

Administrative

- No lecture next Monday on 9/23/2024. I will post some readings though.

- No office hours this week on Friday from me (9/20/204
- Week 03 will still come out this upcoming Friday:
  - DMA, FIR Filter, additional structure in testbenching/verification

- Week 02 stuff is out:
  - More Cocotb
  - More Pynq:
    - Update to the code on the site for last part…

Week 2 Lab:
AXI-Lite Packager Broke

- Still not sure *what* broke going from 2023.2 to 2024.1

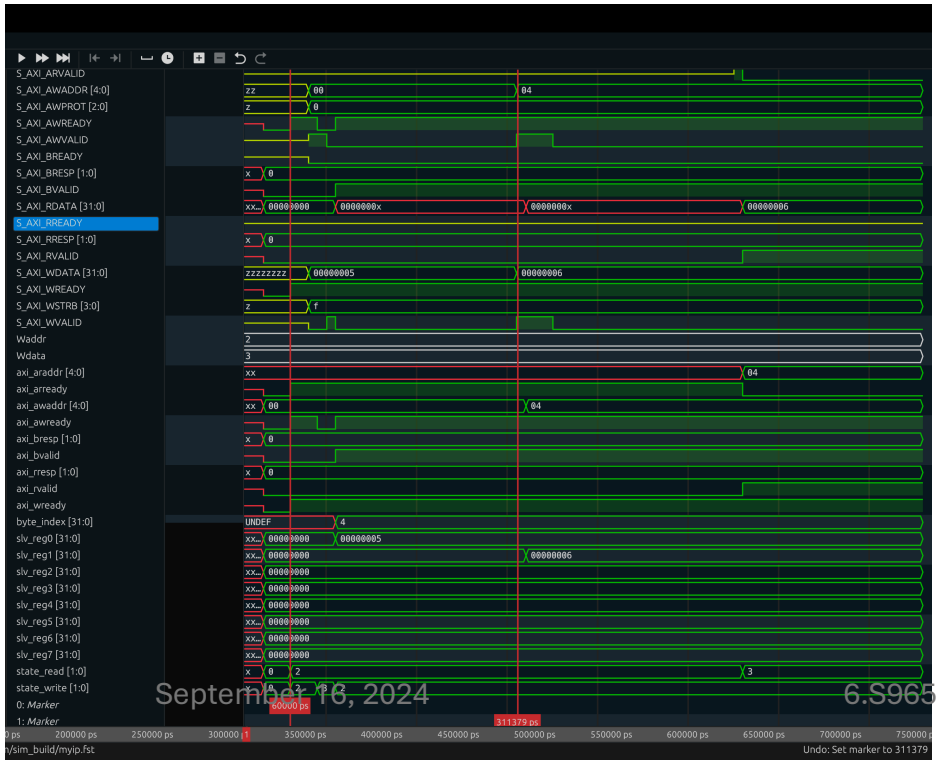
- Didn’t have enough time over the weekend to figure it out

- The new source for AXI Lite mentions burst mode…not sure if that’s a typo or indicative of something else weird.
- Also incompletely specifies read logic compared to <2024.1

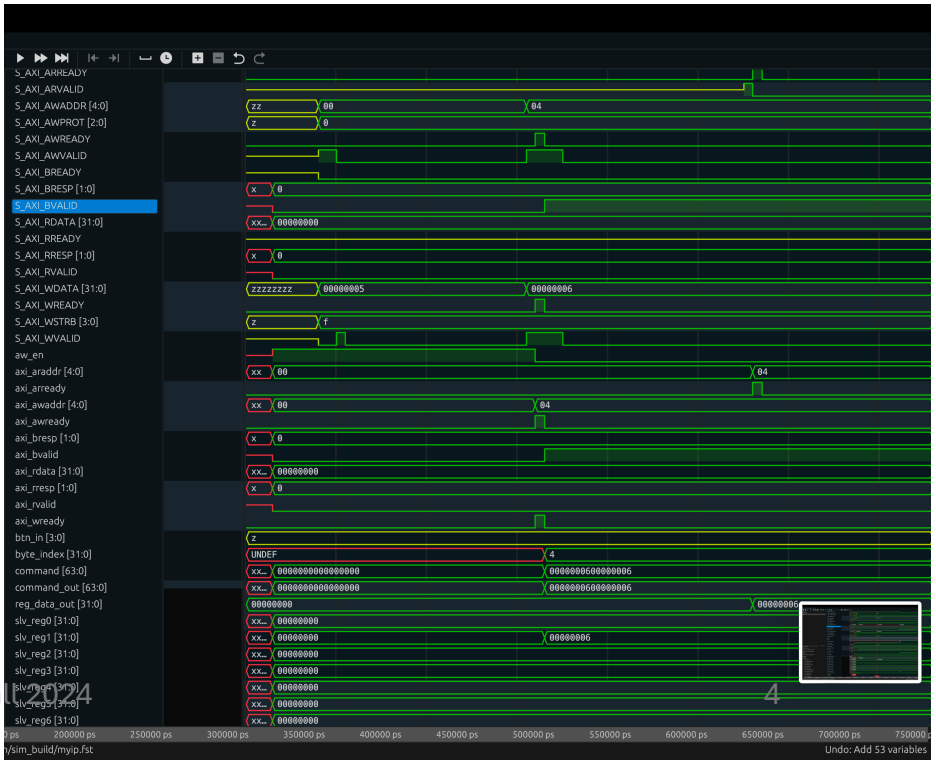
Variants

- I think there's a bug in their READY implementation

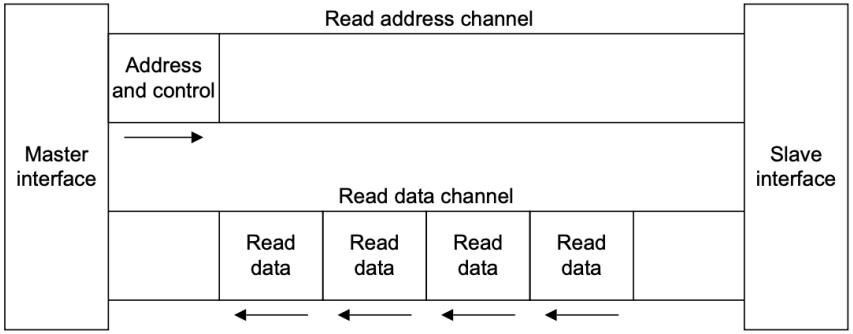
Not-working (2024.1)



Less Not-working (2023.2)

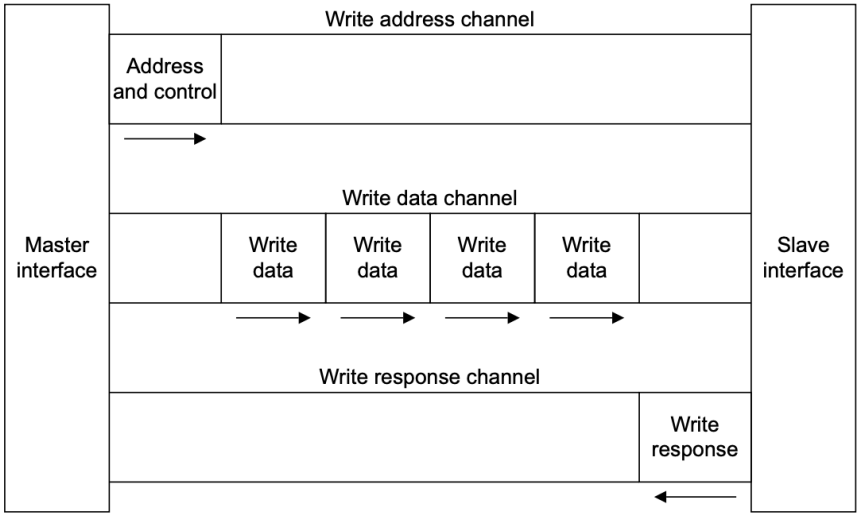


Reads Work Fine



- The “IP wizard” does fail to create all the appropriate read logic by default, but for registers it does, things work
- And you can add in the logic to read the ”forgotten” registers (>4) and things still work

What's Broken?



- Hard Crash/Timeout when a Write is made to the AXI MMIO created
- My guess is it is related to the response channel logic
- An AXI write interface will have three channels:
  - Write Address ("AW") (address to write data to)
  - Write Data ("W") (data to write)
  - Response Data ("B") A response

Generalized Transaction

- All Channel Interactions follow same high-level structure

Table A3-1 Transaction channel handshake pairs

Transaction channel	Handshake pair
Write address channel	AWVALID, AWREADY
Write data channel	WVALID, WREADY
Write response channel	BVALID, BREADY
Read address channel	ARVALID, ARREADY
Read data channel	RVALID, RREADY

Sending One ”beat” of data (one clock-cycle of data)

Keep in mind this could be 64 parallel wires of 1’s and 0’s of info or 8 bytes for example…
Or it could be something else

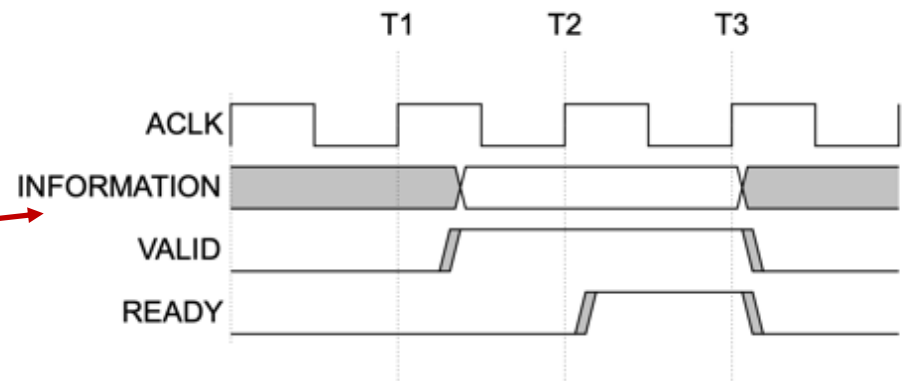


Figure A3-2 VALID before READY handshake

Generalized Transaction

- All Channel Interactions follow same high-level structure

Table A3-1 Transaction channel handshake pairs

Transaction channel	Handshake pair
Write address channel	AWVALID, AWREADY
Write data channel	WVALID, WREADY
Write response channel	BVALID, BREADY
Read address channel	ARVALID, ARREADY
Read data channel	RVALID, RREADY

Sending One ”beat” of data (one clock-cycle of data)

Keep in mind this could be 64 parallel wires of 1’s and 0’s of info or 8 bytes for example…
Or it could be something else

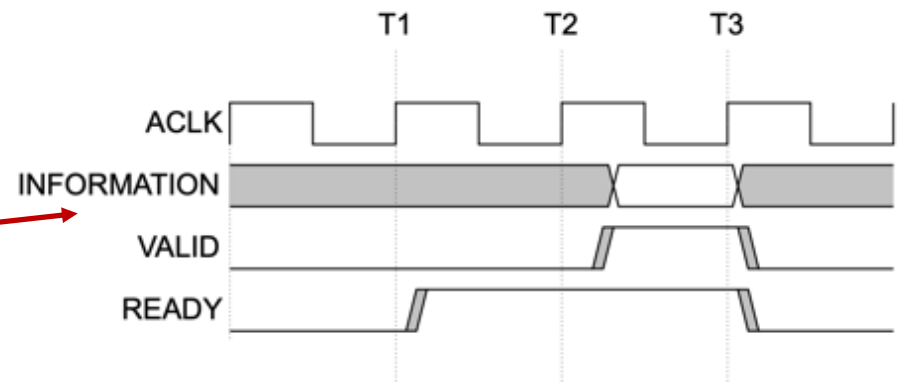


Figure A3-3 READY before VALID handshake

Generalized Transaction

- All Channel Interactions follow same high-level structure

Table A3-1 Transaction channel handshake pairs

Transaction channel	Handshake pair
Write address channel	AWVALID, AWREADY
Write data channel	WVALID, WREADY
Write response channel	BVALID, BREADY
Read address channel	ARVALID, ARREADY
Read data channel	RVALID, RREADY

Sending One ”beat” of data (one clock-cycle of data)

Keep in mind this could be 64 parallel wires of 1’s and 0’s of info or 8 bytes for example…
Or it could be something else

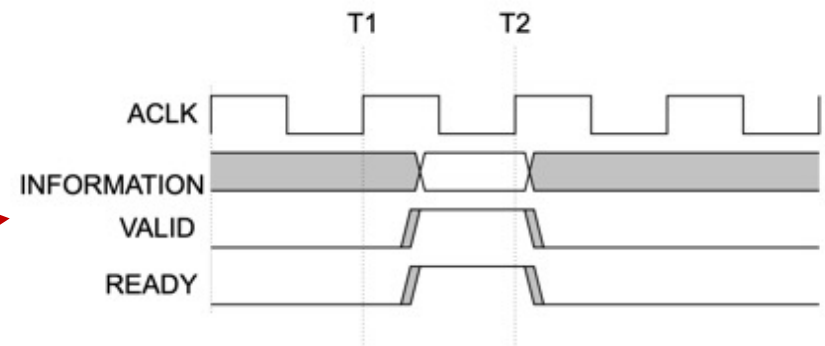


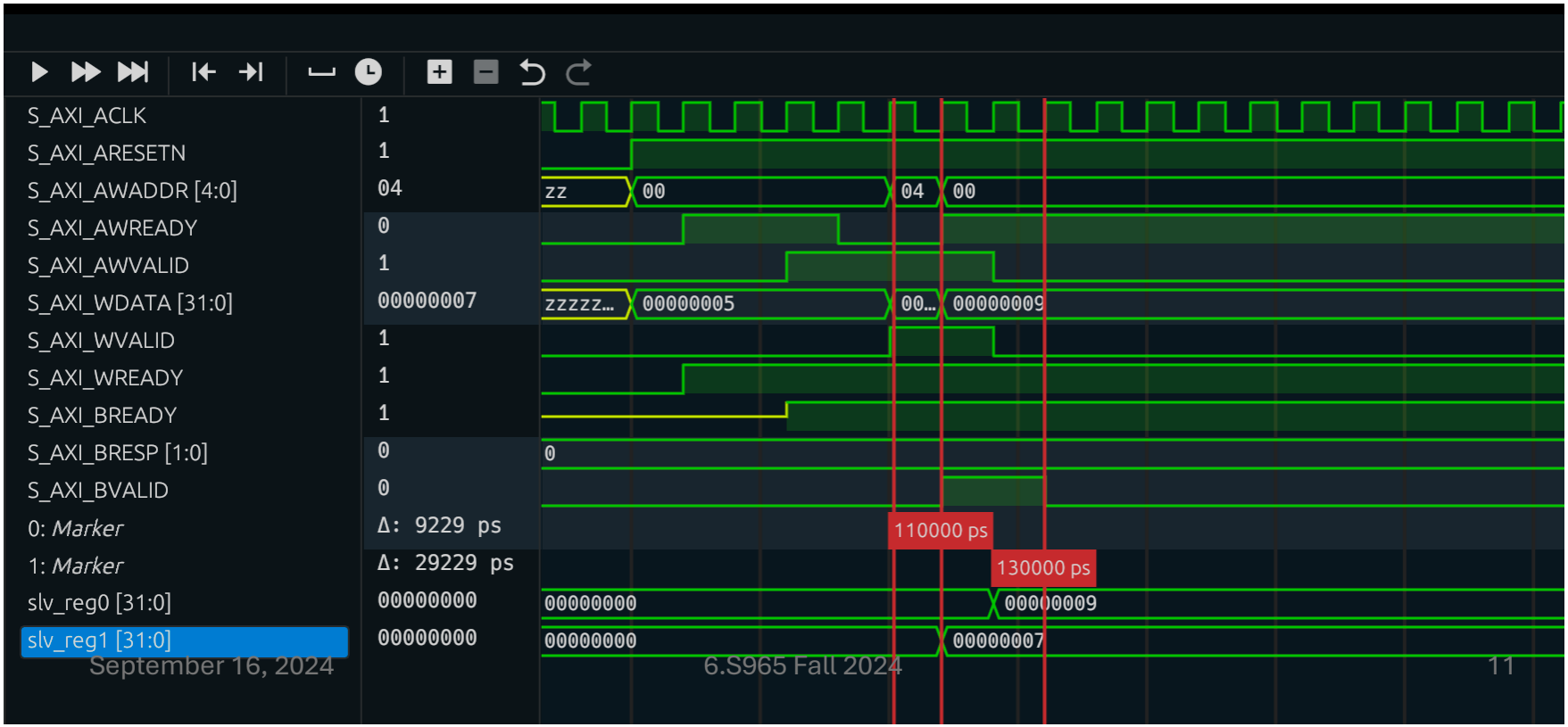
Figure A3-4 VALID with READY handshake

Other Things to Keep in Mind

- the **VALID** signal of the AXI interface sending information *must not be* dependent on the **READY** signal of the AXI interface receiving that information
- an AXI interface that is receiving information *can wait* until it detects a **VALID** signal before it asserts its corresponding **READY** signal.
- Fail to Follow these rules and could have devices wait infinitely.
  - Like when two people keep going “no, after you at a door”

Update: I think this is one issue:

- AXI_AWADDR getting used when AXI_AWREADY and AXI_AWVALID are both not asserted.



Line Letting Un-hand-shaken data through:

- Early in module there is this:

```
if (S_AXI_AWVALID &&    S_AXI_AWREADY)begin
   axi_awaddr <= S_AXI_AWADDR;
```

- Elsewhere the write logic had this

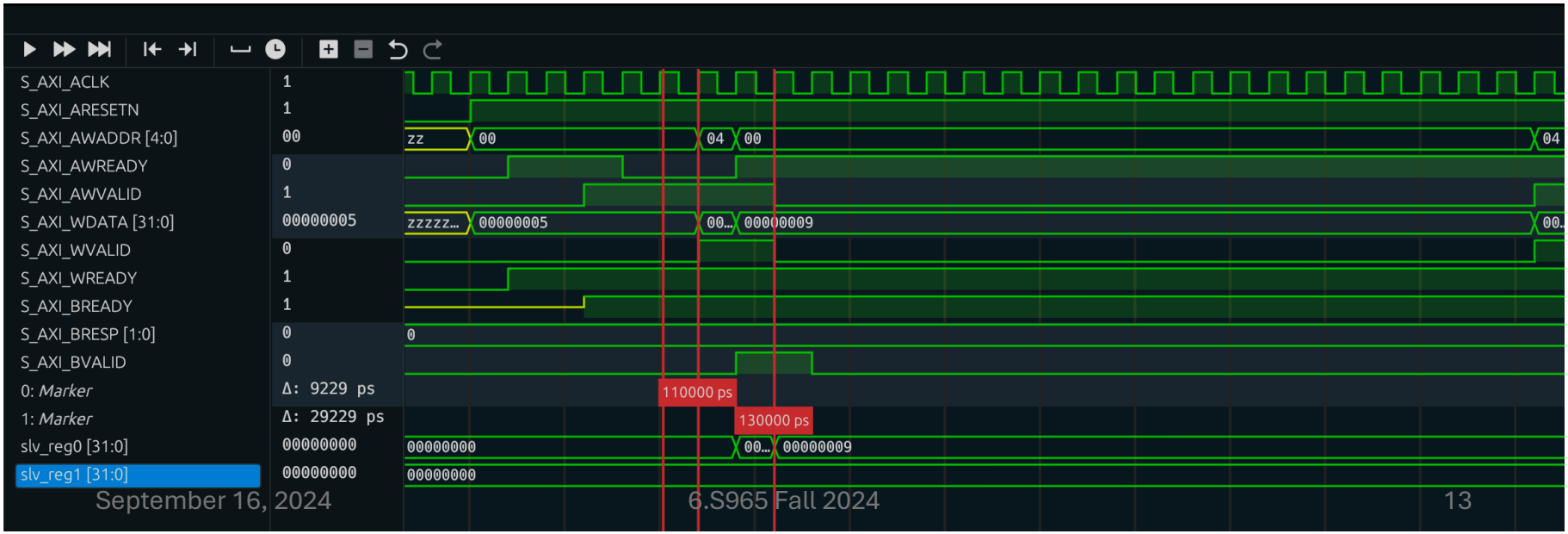
```
//suck:
case ( (S_AXI_AWVALID) ? S_AXI_AWADDR[ADDR_LSB+OPT_MEM_ADDR_BITS:ADDR_LSB]
                       : axi_awaddr[ADDR_LSB+OPT_MEM_ADDR_BITS:ADDR_LSB] )
```

- Change to this:

```
//seems better:
case ( (S_AXI_AWREADY && S_AXI_AWVALID) ? S_AXI_AWADDR[ADDR_LSB+OPT_MEM_ADDR_BITS:ADDR_LSB]
                                        : axi_awaddr[ADDR_LSB+OPT_MEM_ADDR_BITS:ADDR_LSB] )
```

Update: I think this is part of issue…

- There's a line that uses the raw address based only on AXI_AWVALID
- Change it



Still Not Exactly Sure

- So haven't tested it. But that is at least one "hole" in the spec that the other, older module does not fall prey to.
- Anyways, I'll keep looking. This might form part of week 3's assignment
- This is part of a larger issue. A lot of Xilinx stuff and many vendors is "AXI-ish"…fails on some edge cases. This can get very frustrating when using encrypted IP

- Also I'm not the first to talk about this

AXI Culture

- This Gisselquist guy is anywhere anybody mentions AXI on the internet

zipcpu.com/blog/2021/05/22/vhdlaxil.html

Gisselquist Technology, LLC

Main/Blog
About Us
FPGA Hell
Tutorial

Fixing Xilinx's Broken AXI-lite Design in VHDL

ZipCPU OP • 5y ago

> I think you have some basic misconception of what AXI actually is.

I'm willing to believe I have such a basic misconception. This is why I'm writing and asking for enlightenment. Thank you for taking the time to help me understand this here.

> It's a high performance protocol.

This may be where I need the most enlightenment. To me, a "high performance protocol" is one that allows one beat of information to be communicated on every clock. Many if not most of the AXI implementations I've seen don't actually hit this target simply because all of the extra logic required to implement the bus slows it down. There's also something to be said for low-latency, but in general my biggest criticisms are of lost throughput.

> You can take advantage of slave features like command reordering with DDR.

While I have filed bug reports in 2017 and 2018 on Xilinx's forums regarding these broken demonstration designs, Xilinx has yet to fix their designs as of Vivado 2020.2. [1], [2] Indeed, at this point, it's not clear if Xilinx will ever fix their demonstration designs. Perhaps I shouldn't complain–their designs simply make the services I offer and sell that much more valuable.

The most common AXI mistake

Apr 16, 2019

Some time ago, I posted a set of formal properties which could slave or master. I then applied these properties to the AXI-lite sla and found multiple errors within their core.

D@n

Members

2.3k 32

Posted January 8, 2020

@HasanWAVE,

Might it be because your Zynq design only supports AXI3 and not AXI4? The maximum AXI3 burst length is only 16 beats.

Which board are you using?

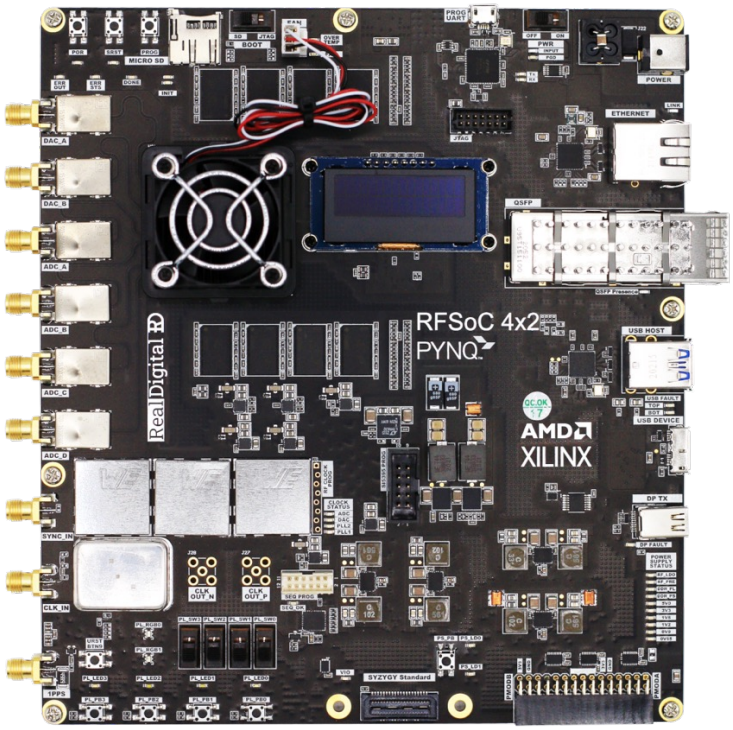
Also, this really belongs in the FPGA/embedded forum, not the microcontroller forum.

Dan

Signal Processing on the FPGA

6.S965 RFSoC

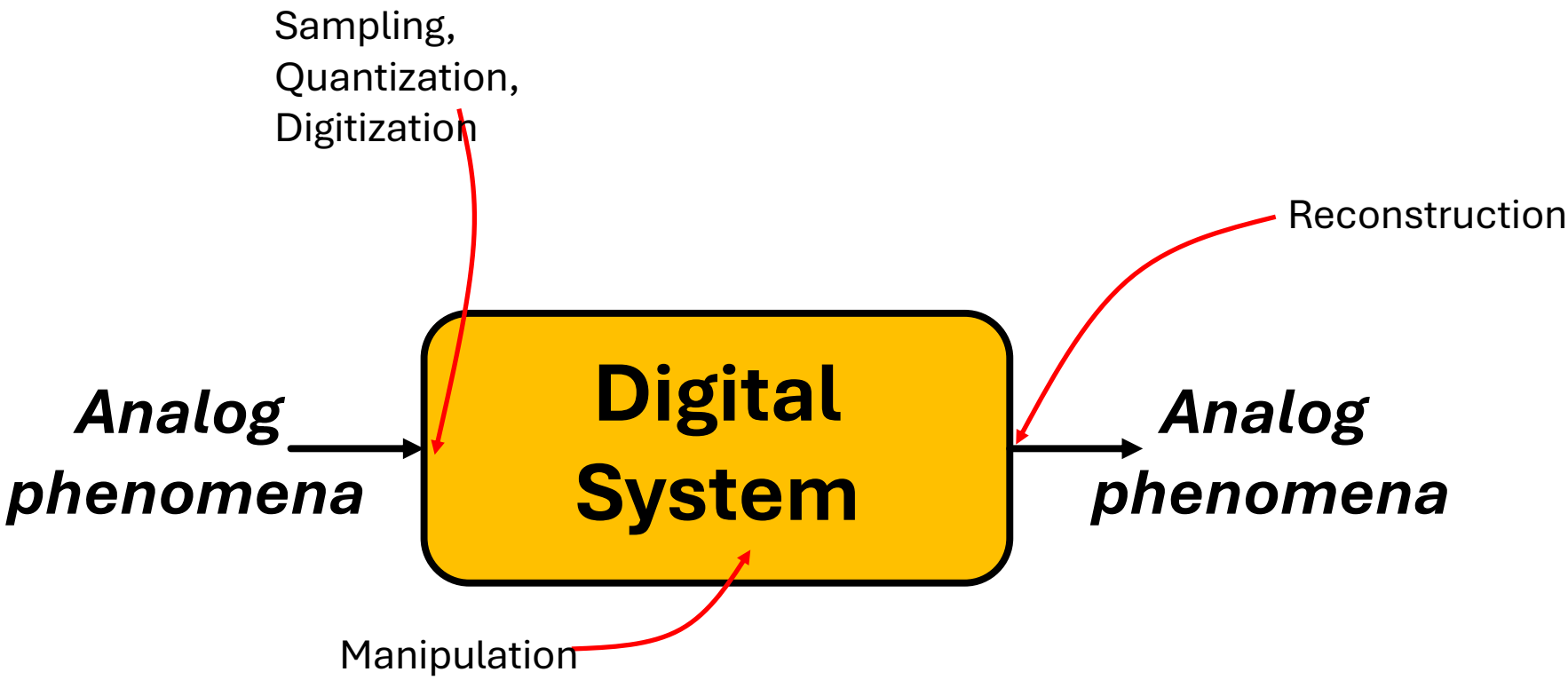
- UltraScale+ ZU48DR:
  - 38 Mb of BRAM
  - +22Mb of UltraRAM
  - 4272 DSP slices
  - 930,000 Logic Cells
  - **Four 5-Gsps 14 bit ADCs**
  - **Two 10-Gsps 14 bit DACs**
  - Four 1.3 GHz ARM 53 processors
  - Two Real-time 533 MHz ARM processors
- Board has 4GB of DDR4 for FPGA portion (”PL”) and 4 GB of DDR4 for processors (“PS”)



https://www.amd.com/en/products/adaptive-socs-and-fpgas/soc/zynq-ultrascale-plus-rfsoc.html#tabs-b3ecea84f1-item-e96607e53b-tab

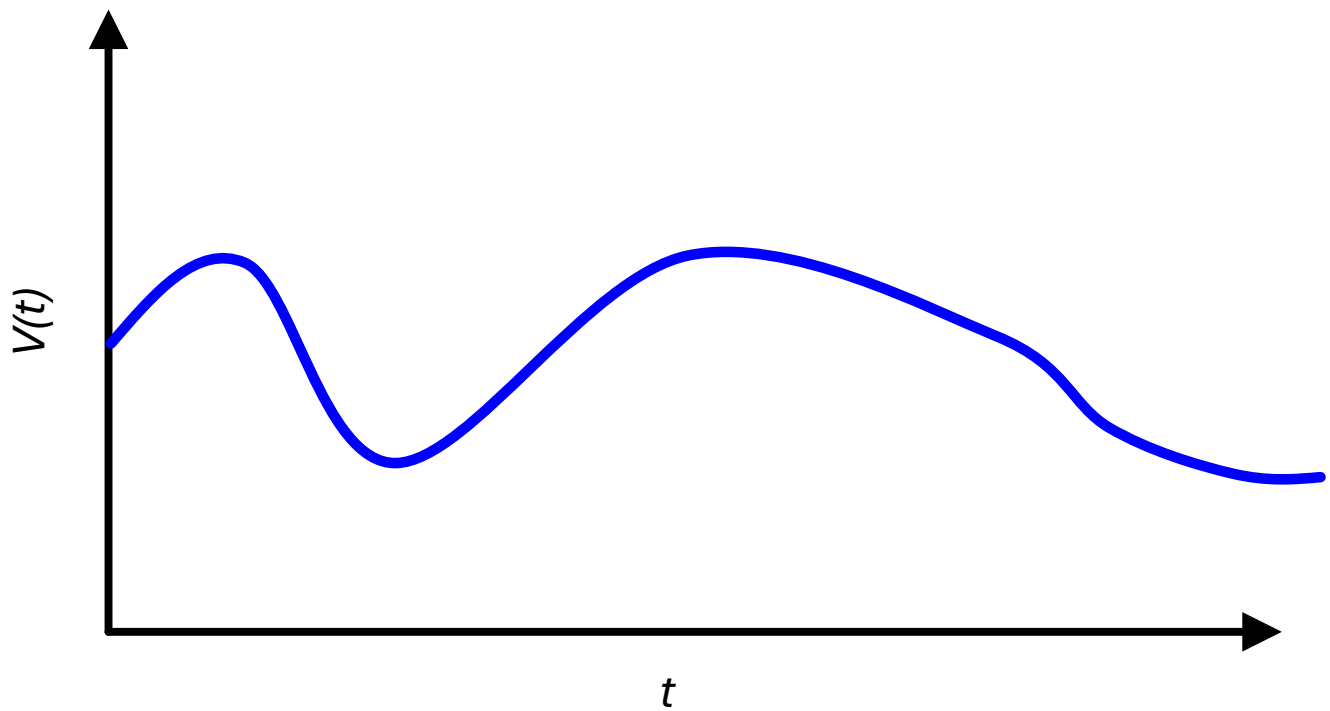
A Digital System in an Analog World

- Many physical phenomena (sound, light, physics in general) are best-described as continuous entities

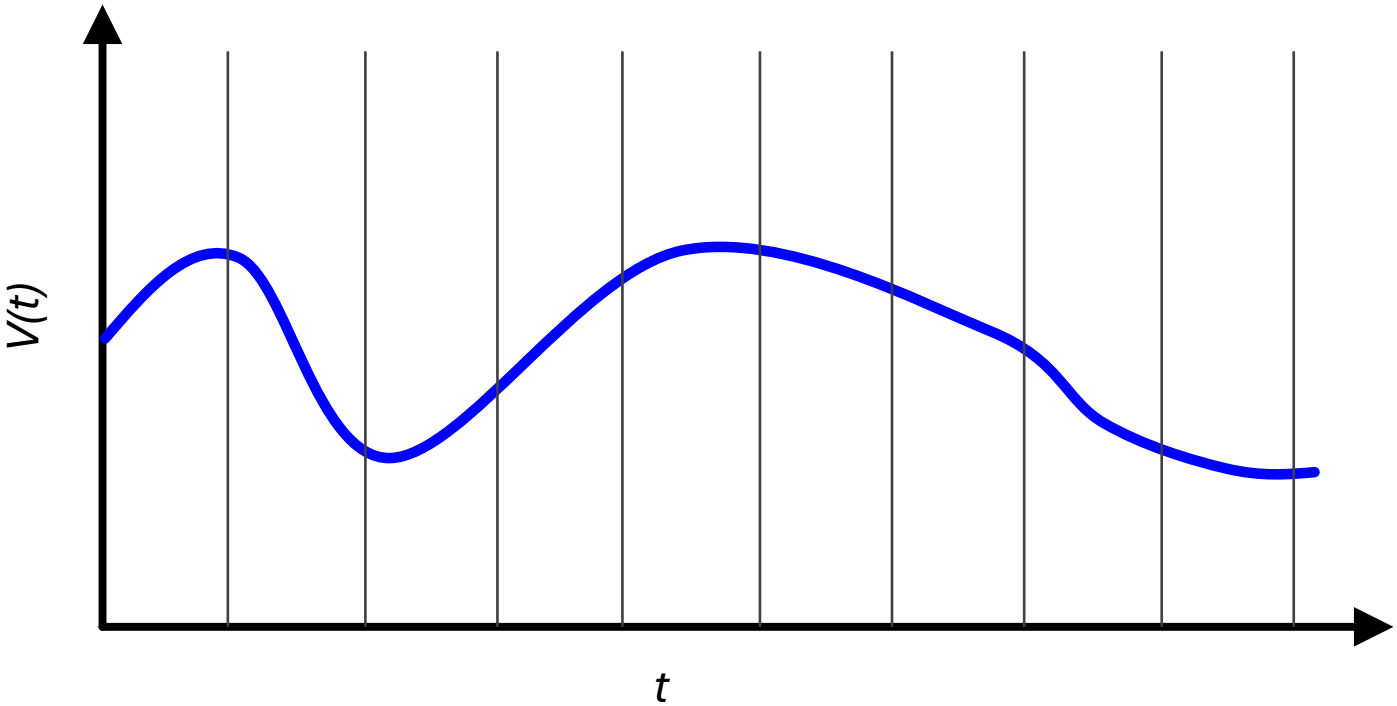


Visualizing Sampling

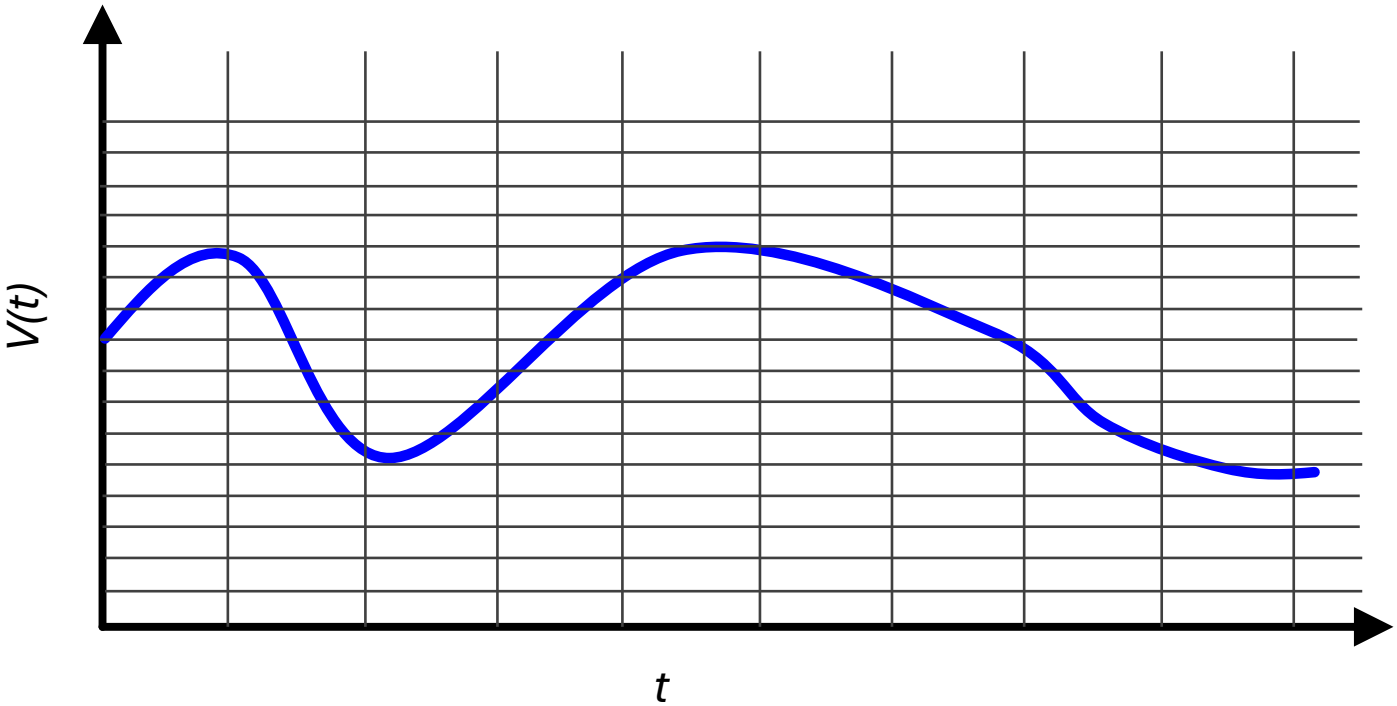
Continuous in Value and in Time



Discretization in Time

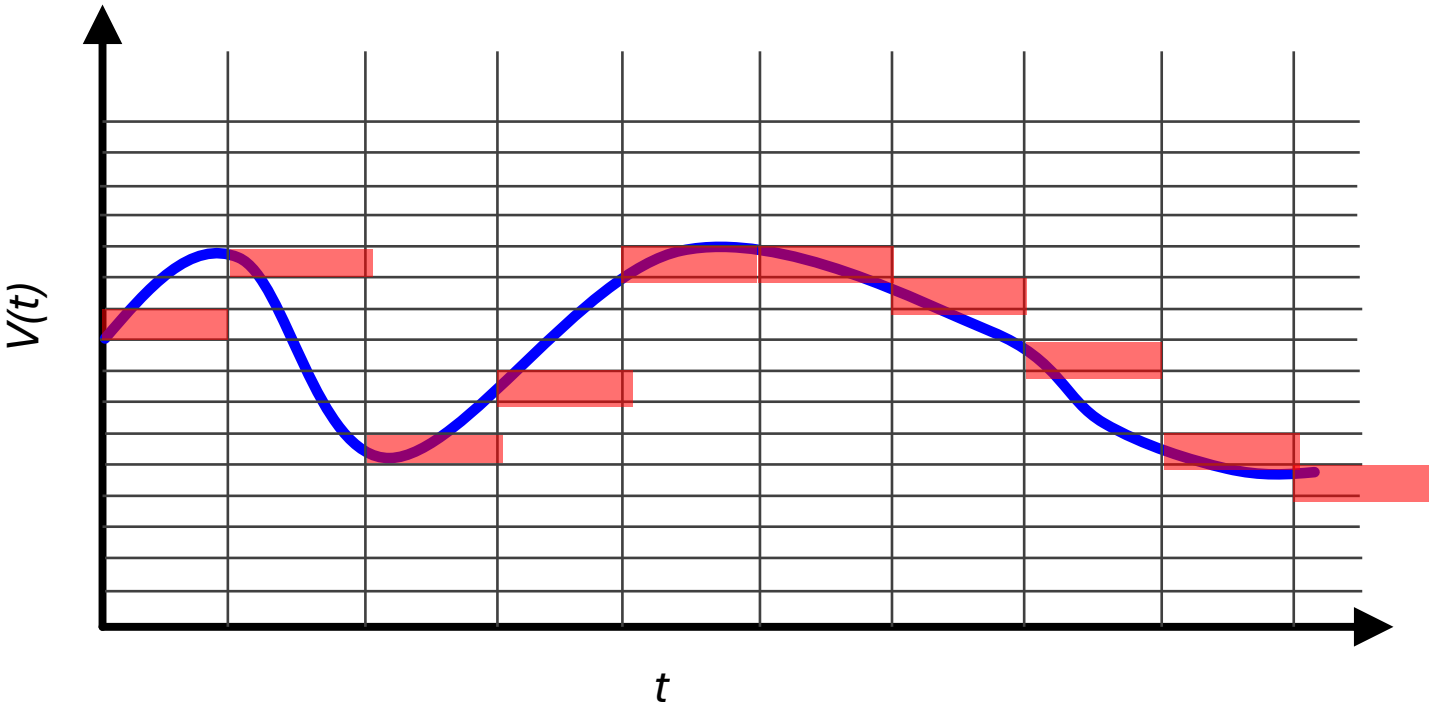


Discretization in Time and **Quantization** in Value



4 bit value encoding

Discretization in Time and **Quantization** in Value



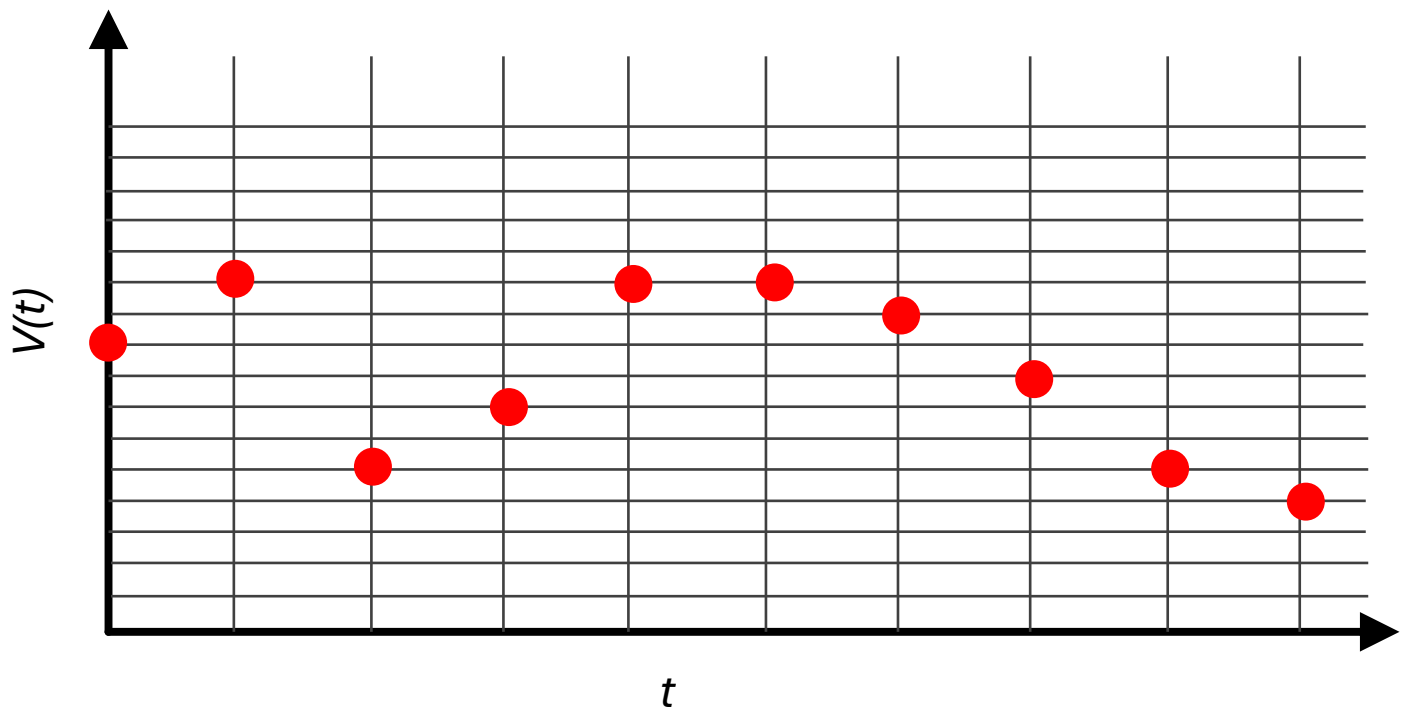
v[n] = [9,11,5,7,11,11,10,8,5,4,]

4 bit value encoding

Store in memory

- v[n] = [9,11,5,7,11,11,10,8,5,4,]
- 10 4-bit values: need 40 bits to represent!
- Good stuff. That’s not a lot!

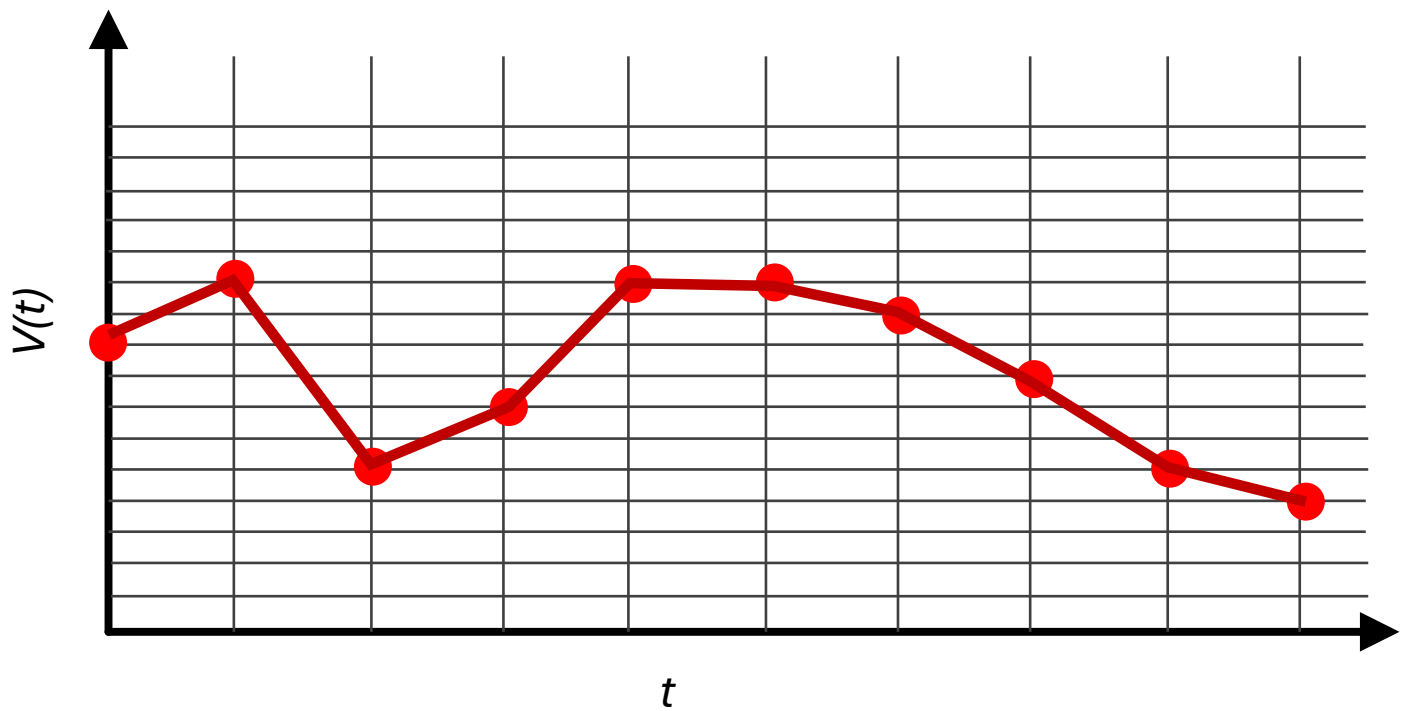
Reconstruction of Signal



v[n] = [9,11,5,7,11,11,10,8,5,4,]

4 bit value encoding

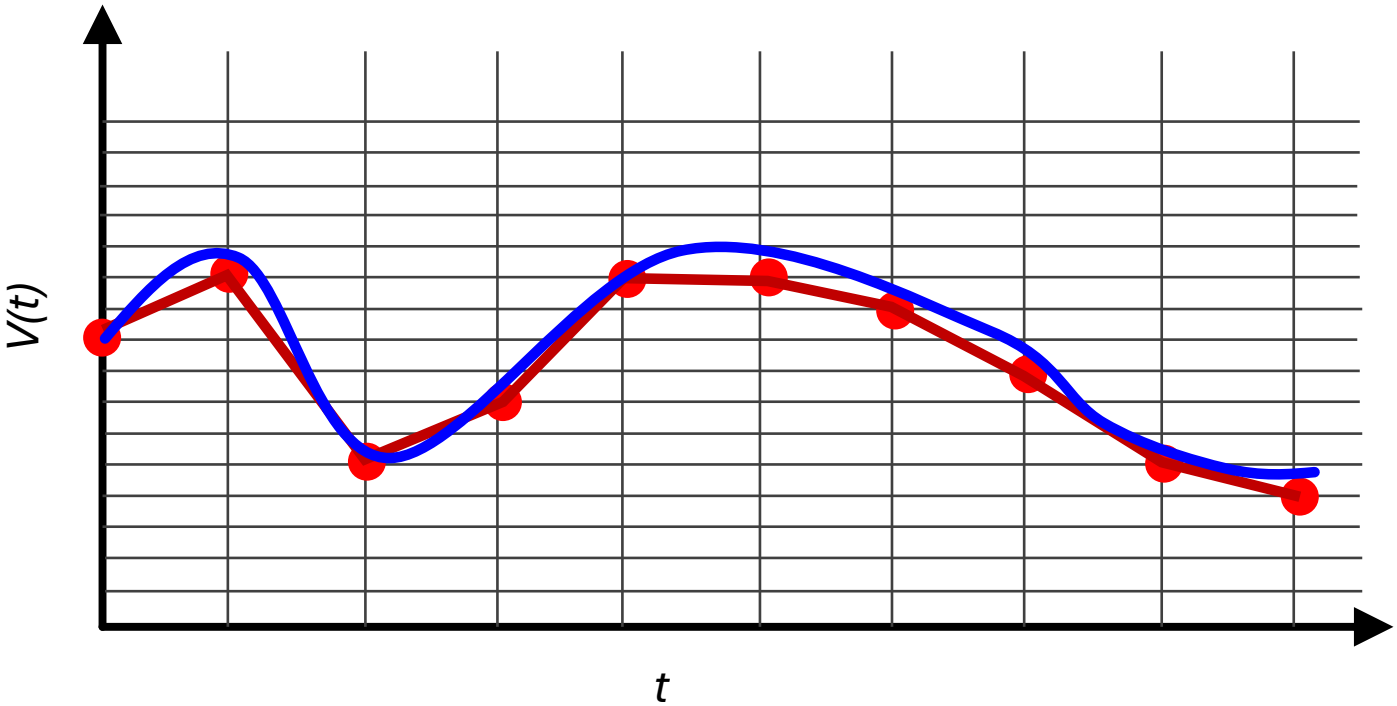
Reconstruction (with first-order hold interpolation)



v[n] = [9,11,5,7,11,11,10,8,5,4,]

4 bit value encoding

Compare to original… not bad



v[n] = [9,11,5,7,11,11,10,8,5,4,]

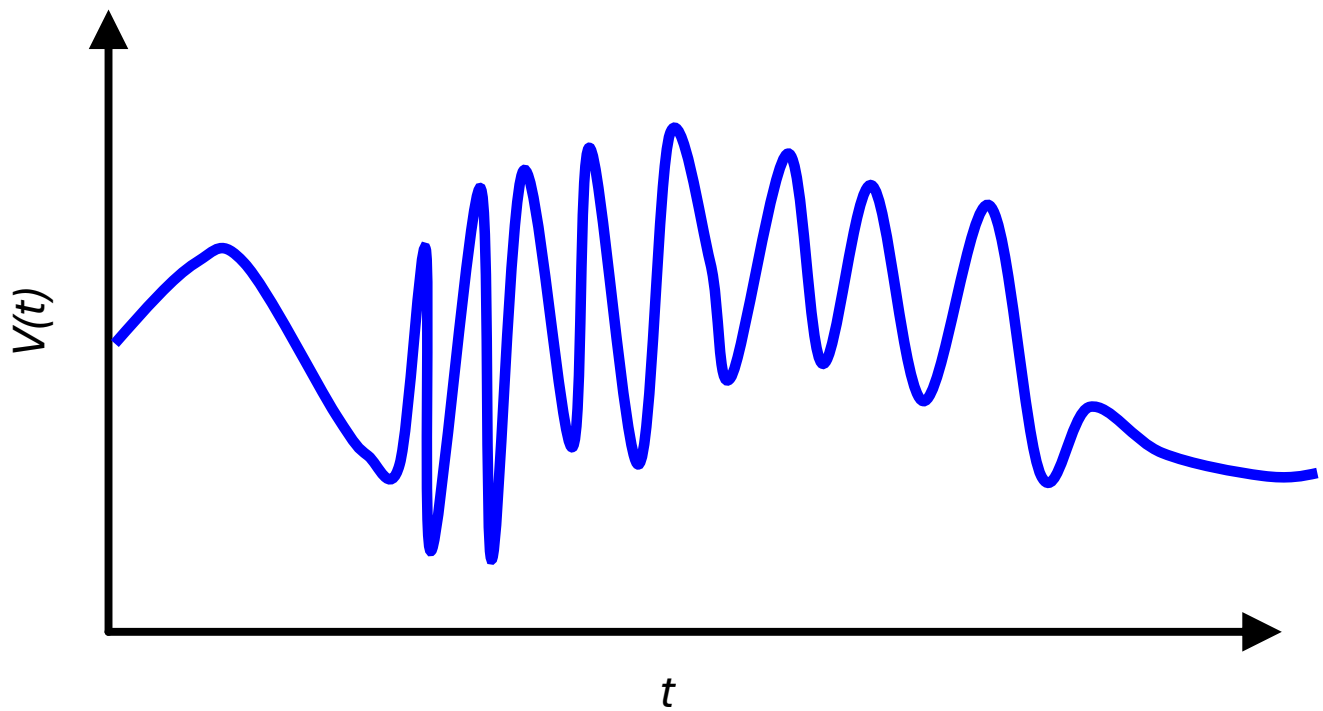
4 bit value encoding

Errors

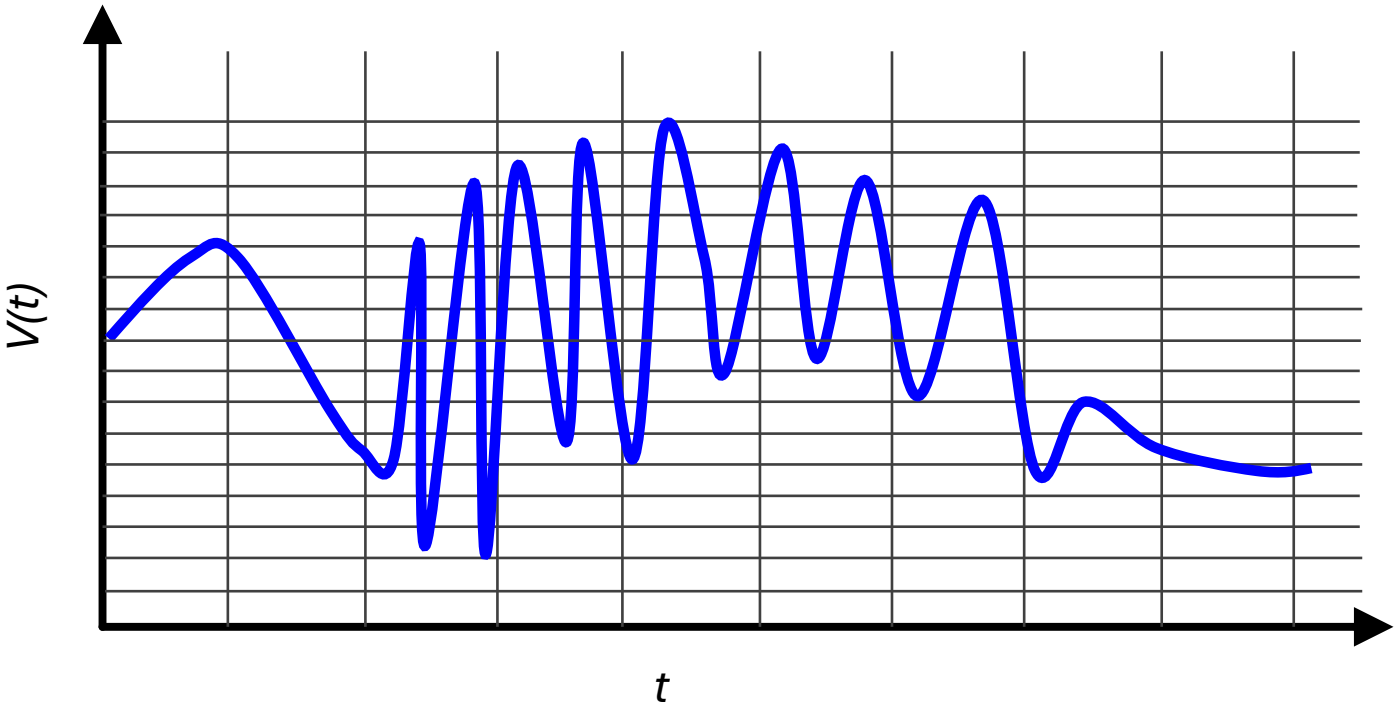
- **Discretization Error:** How ”off” our readings are in time due to sampling at discrete intervals

- **Quantization Error:** How “off” our readings are in reproduced value…if our bin size is 50mV and our signal varies only by 20mV this is going to cause problems

Continuous in Value and in Time

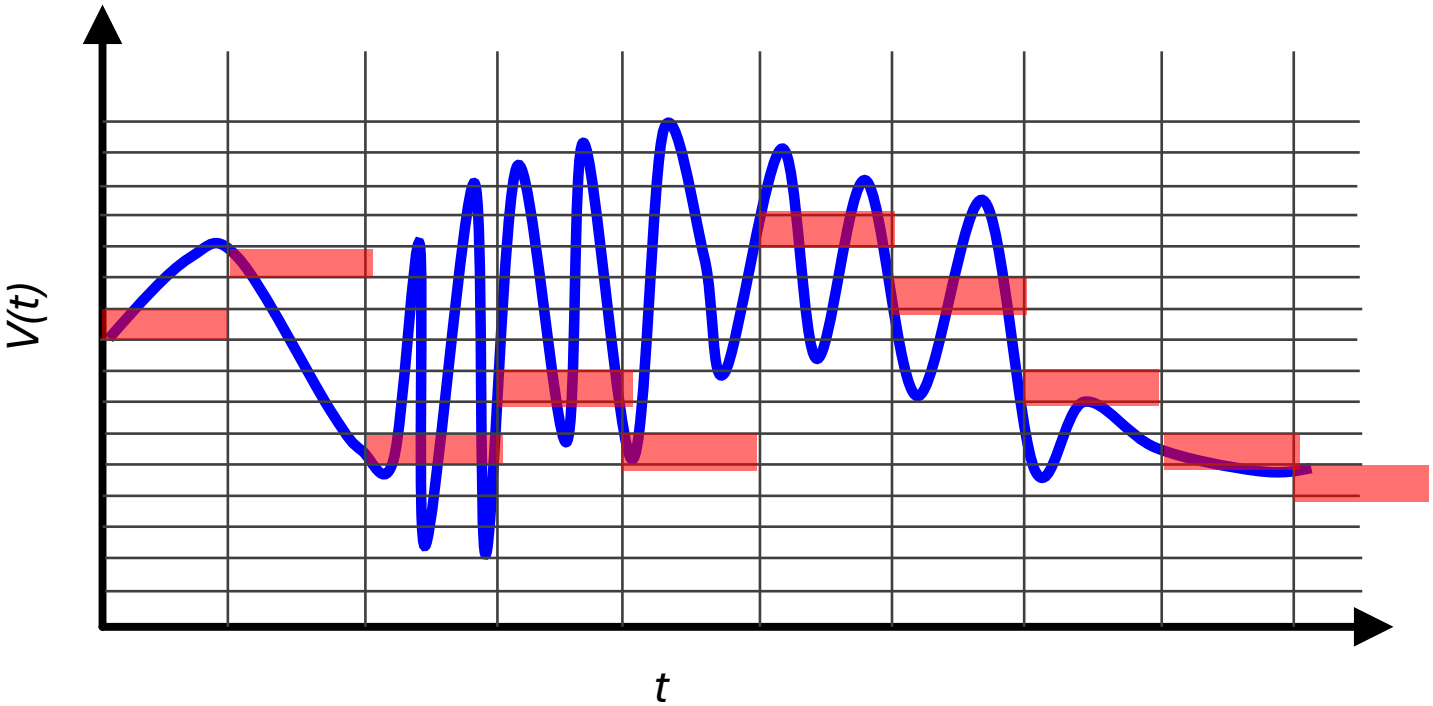


Discretization in Time and **Quantization** in Value



4 bit value encoding

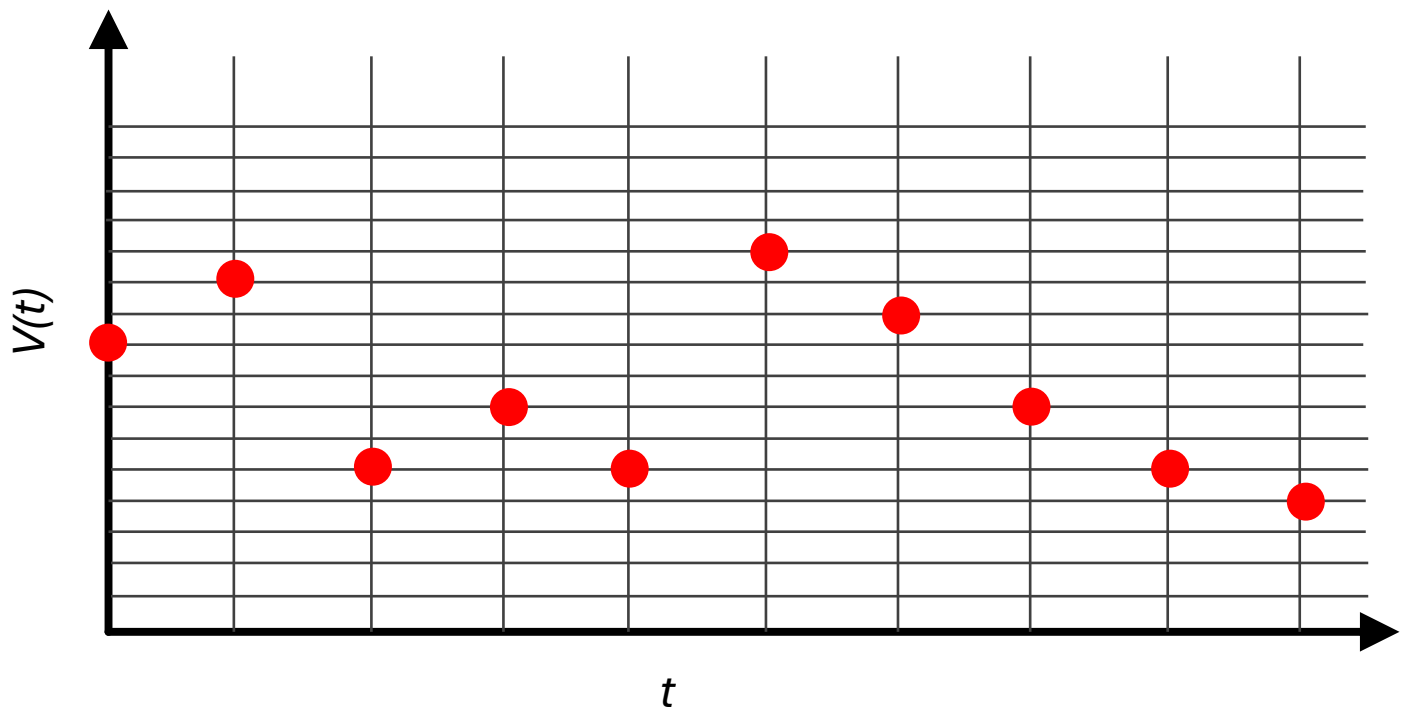
Discretization in Time and **Quantization** in Value



v[n] = [9,11,5,7,5,12,10,7,5,4,]

4 bit value encoding

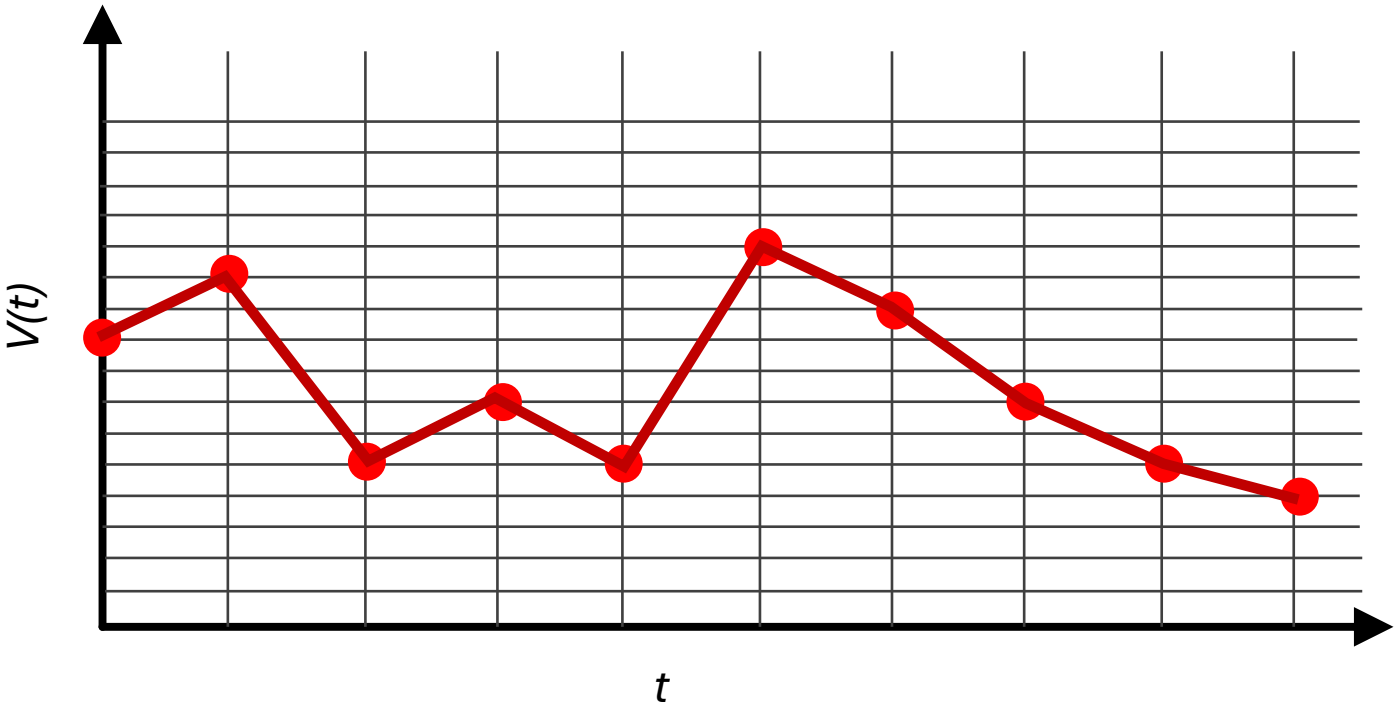
Reproduce



v[n] = [9,11,5,7,5,12,10,7,5,4,]

4 bit value encoding

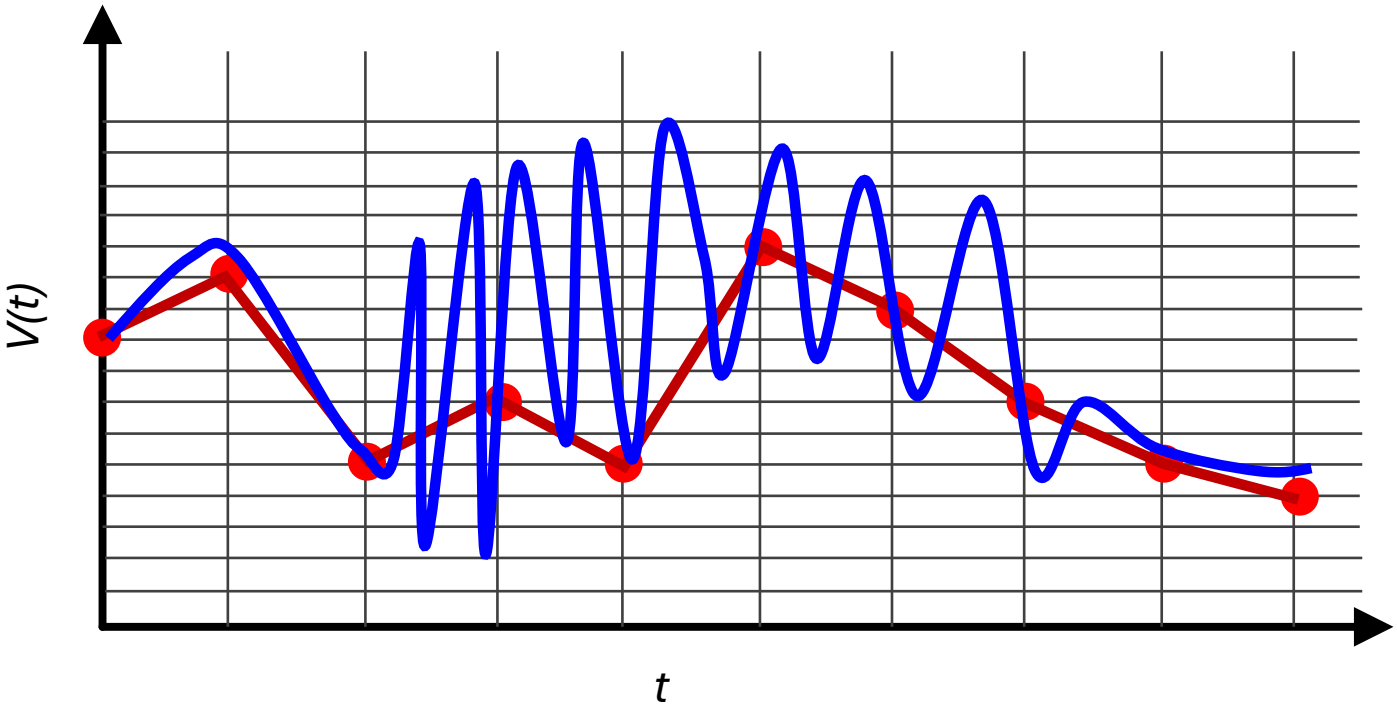
Reproduce



v[n] = [9,11,5,7,5,12,10,7,5,4,]

4 bit value encoding

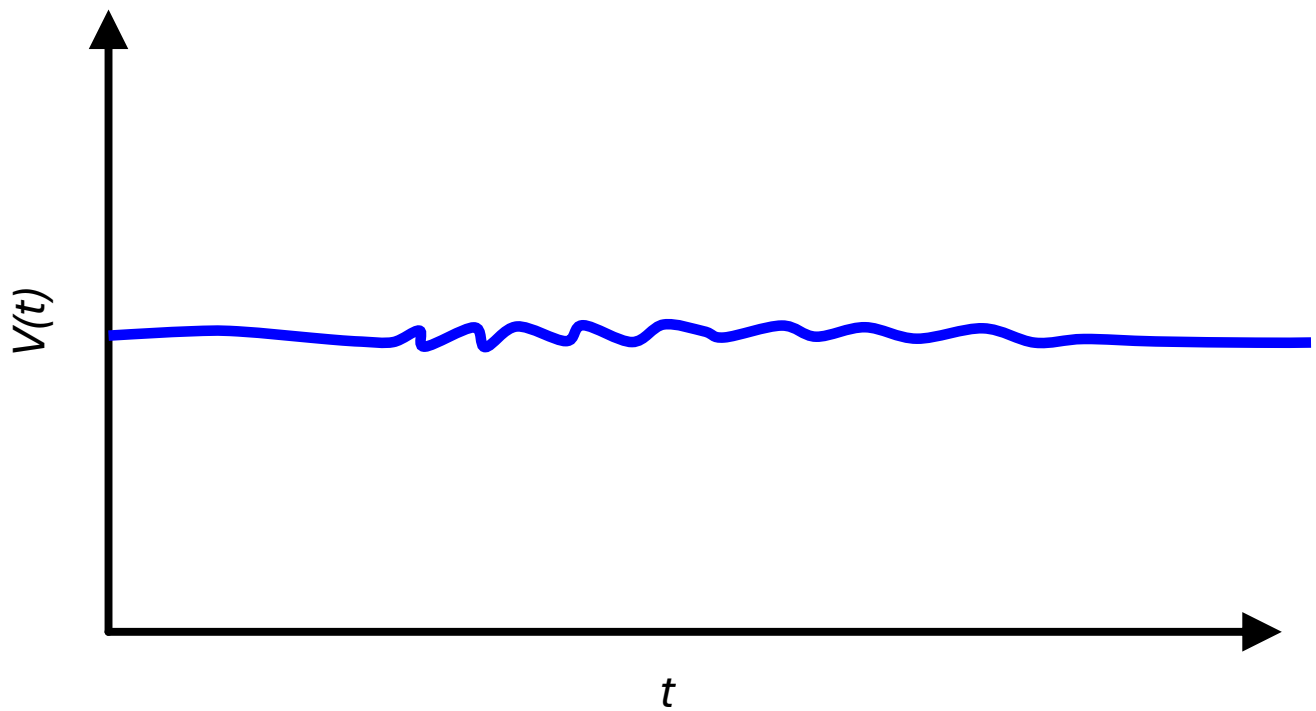
Compare to original… Did not Capture the high-frequency Wiggles!



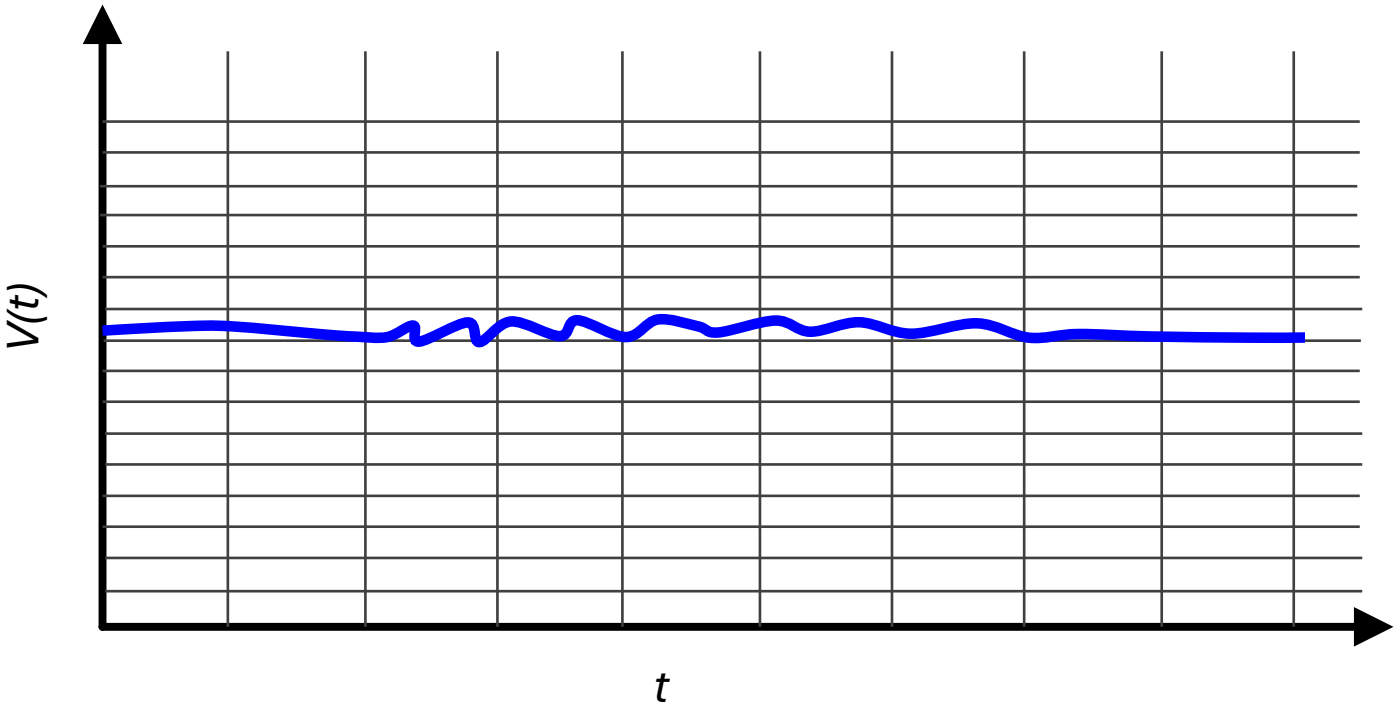
v[n] = [9,11,5,7,5,12,10,7,5,4,]

Potentially Bad Discretization Error

Continuous in Value and in Time

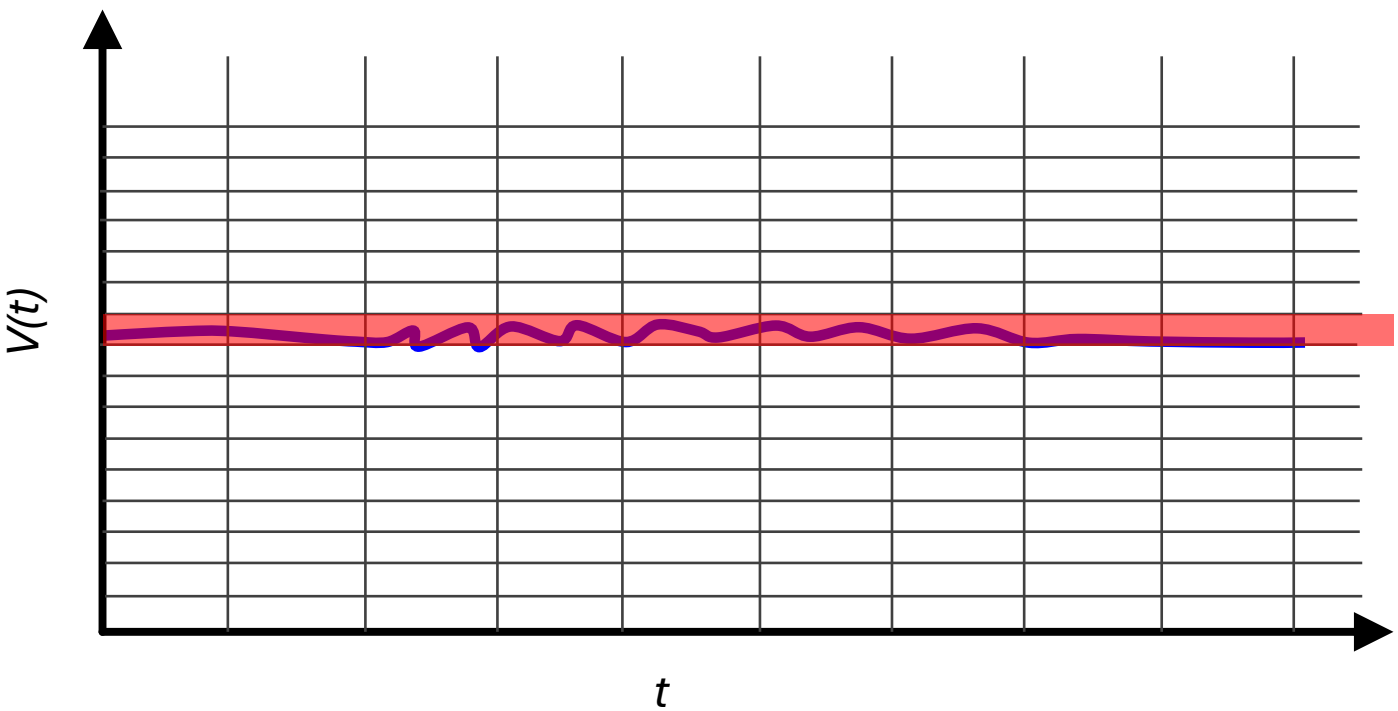


Discretization in Time and **Quantization** in Value



4 bit value encoding

Discretization in Time and **Quantization** in Value



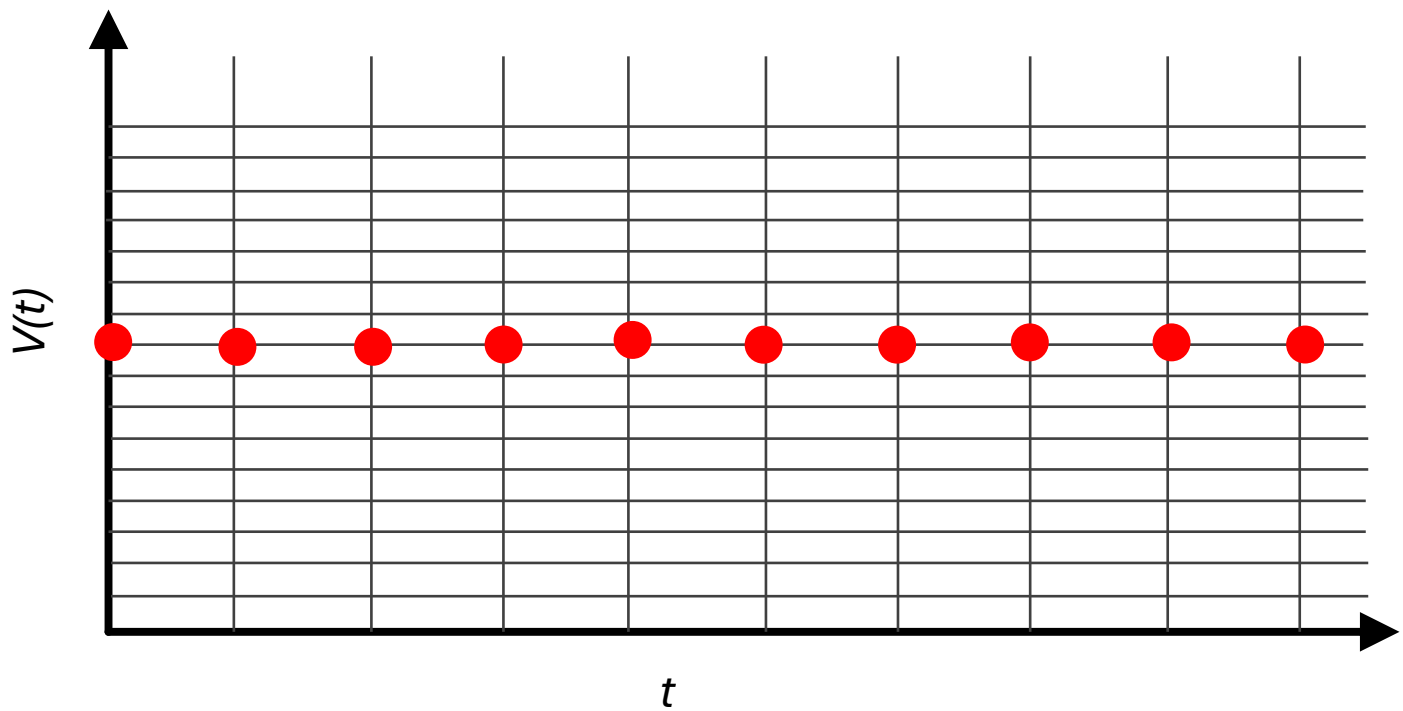
v[n] = [9,9,9,9,9,9,9,9,9,9]

4 bit value encoding

Store in memory

- v[n] = [9,9,9,9,9,9,9,9,9,9]
- 10 4-bit values: need 40 bits in memory!
- Great. All is good.

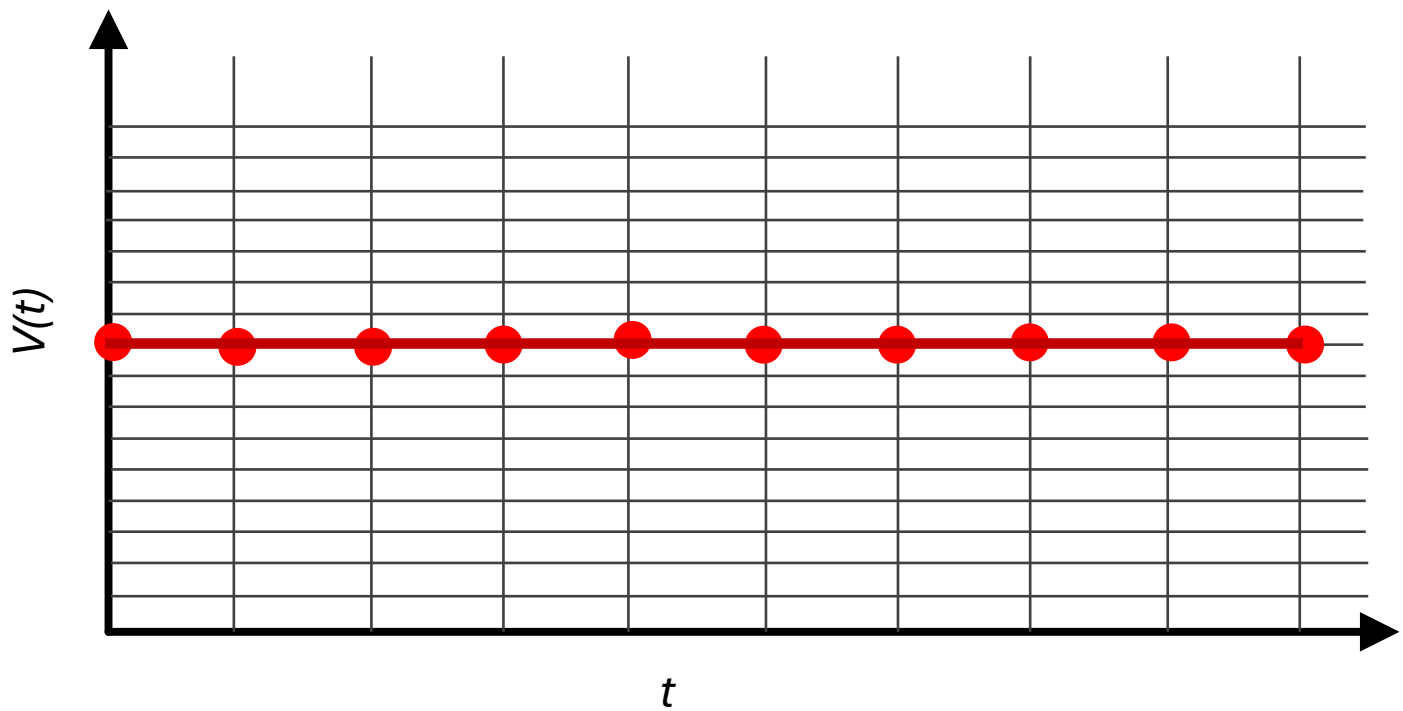
Reproduce



v[n] = [9,9,9,9,9,9,9,9,9,9]

4 bit value encoding

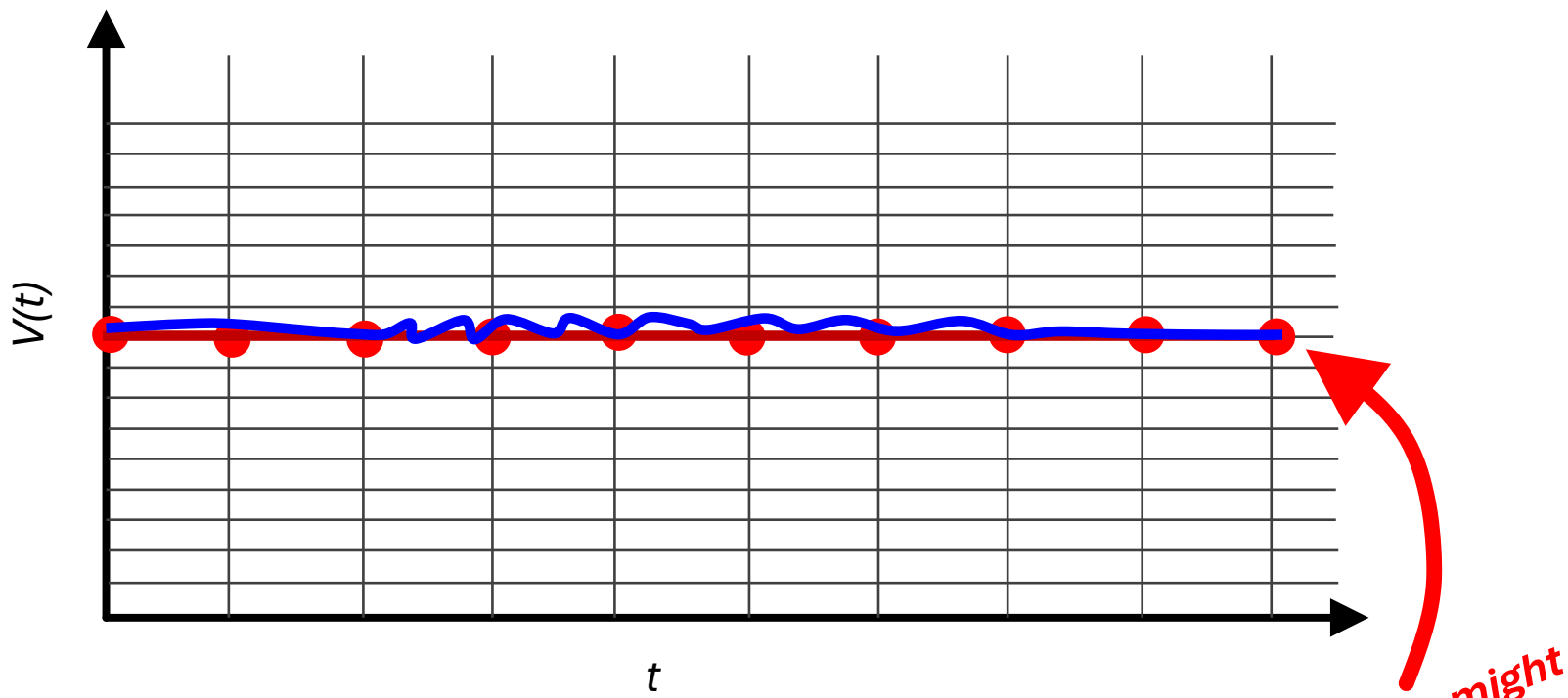
Reproduce



v[n] = [9,9,9,9,9,9,9,9,9,9]

4 bit value encoding

Compare… to original also meh



v[n] = [9,9,9,9,9,9,9,9,9,9]

Potentially Really Problematic

Quantization Error!

Those tiny wiggles might be really important in certain contexts! Tiny heartbeats!

Conclusions

- Care must be taken when choosing what rate you sample (**discretize**) your signal and at what bit-depth you **quantize** your sample
- There's no right answer, since it depends on context/use cases.
- Ideally want to sample at high rate and quantize with many bits…
- But taken to the extreme this uses a lot of resources (lots of memory and resources/lots of bits) so downward pressure on choices

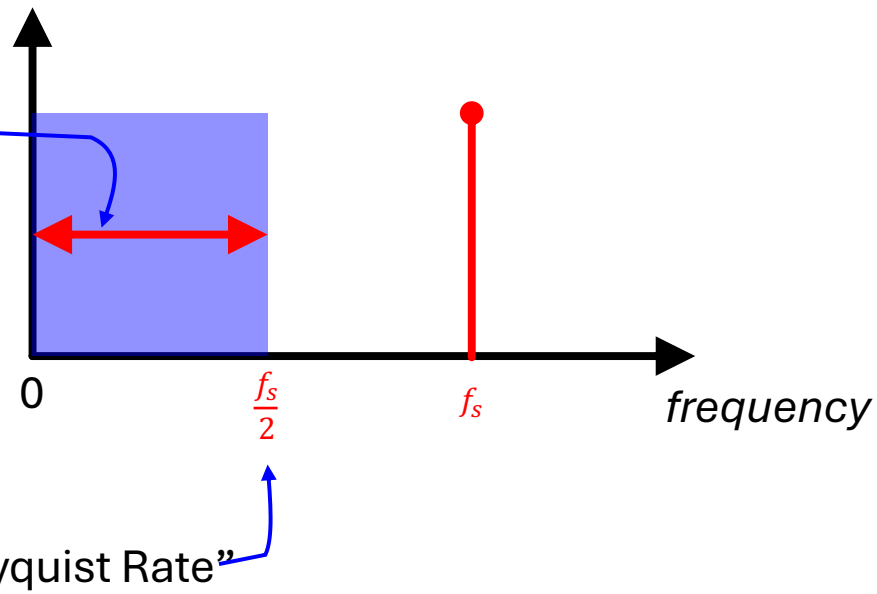
Is that all there is to it?

- No, it is wayyy more complicated
- Let's just consider sample rate for right now (we'll revisit quantization later)

Sample Rate

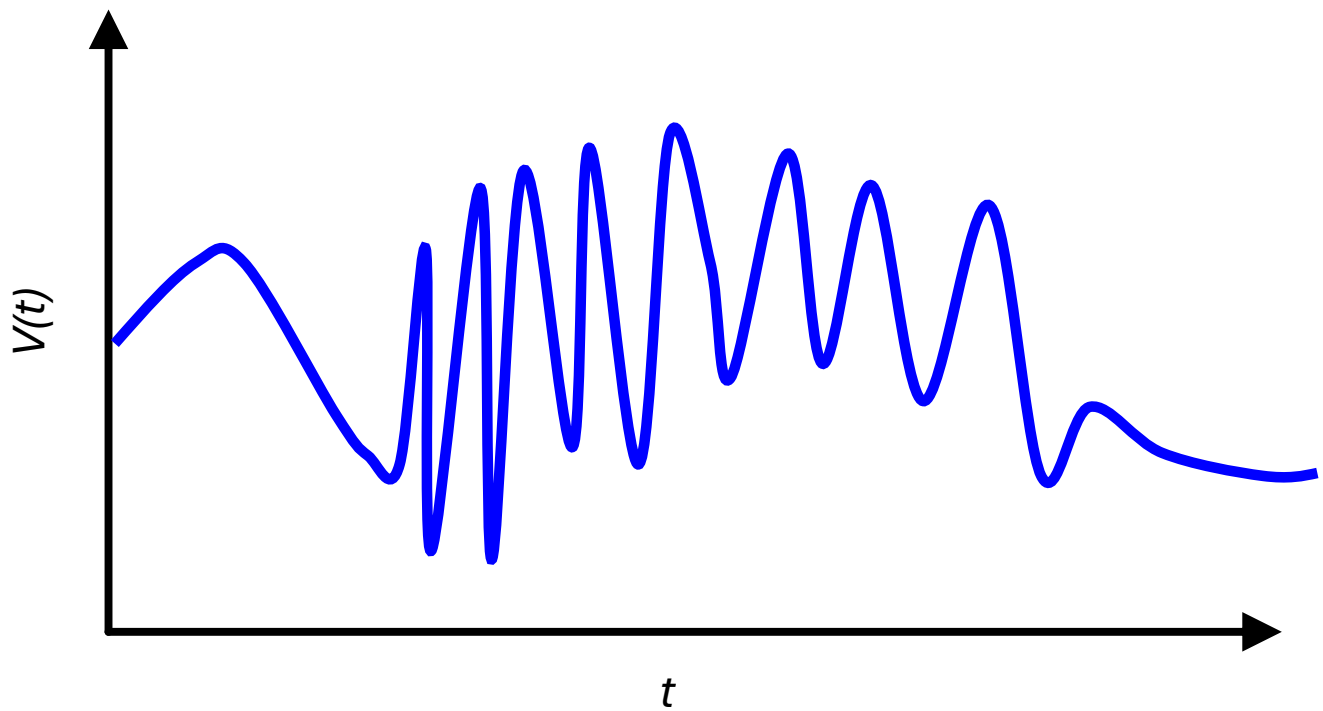
- How frequently we sample our signal directly influences what we can effectively capture.
- A sample rate of f_s is only capable of expressing signals with frequencies less than $\frac{f_s}{2}$

Signals with frequencies in this region of the spectrum can be ***fully captured***

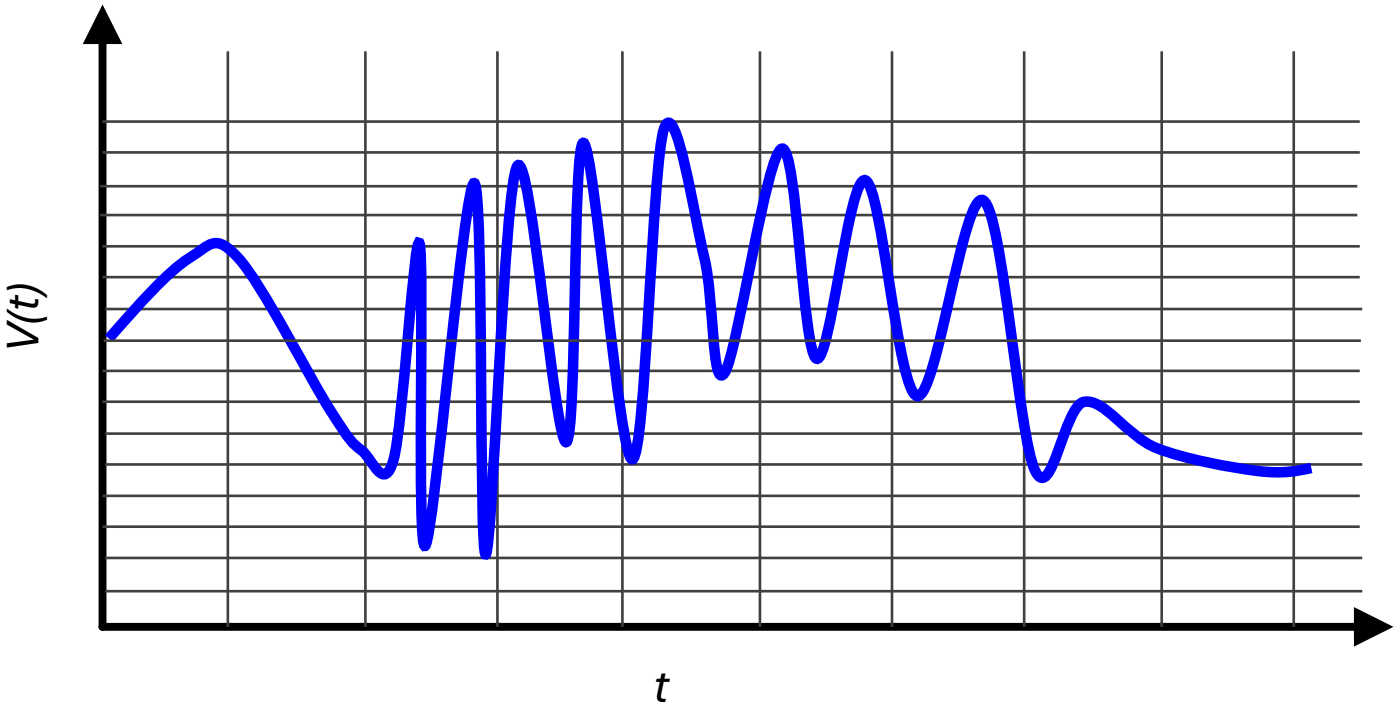


Nyquist, Shannon, few others showed this in the 1930s

Let’s consider this situation though….

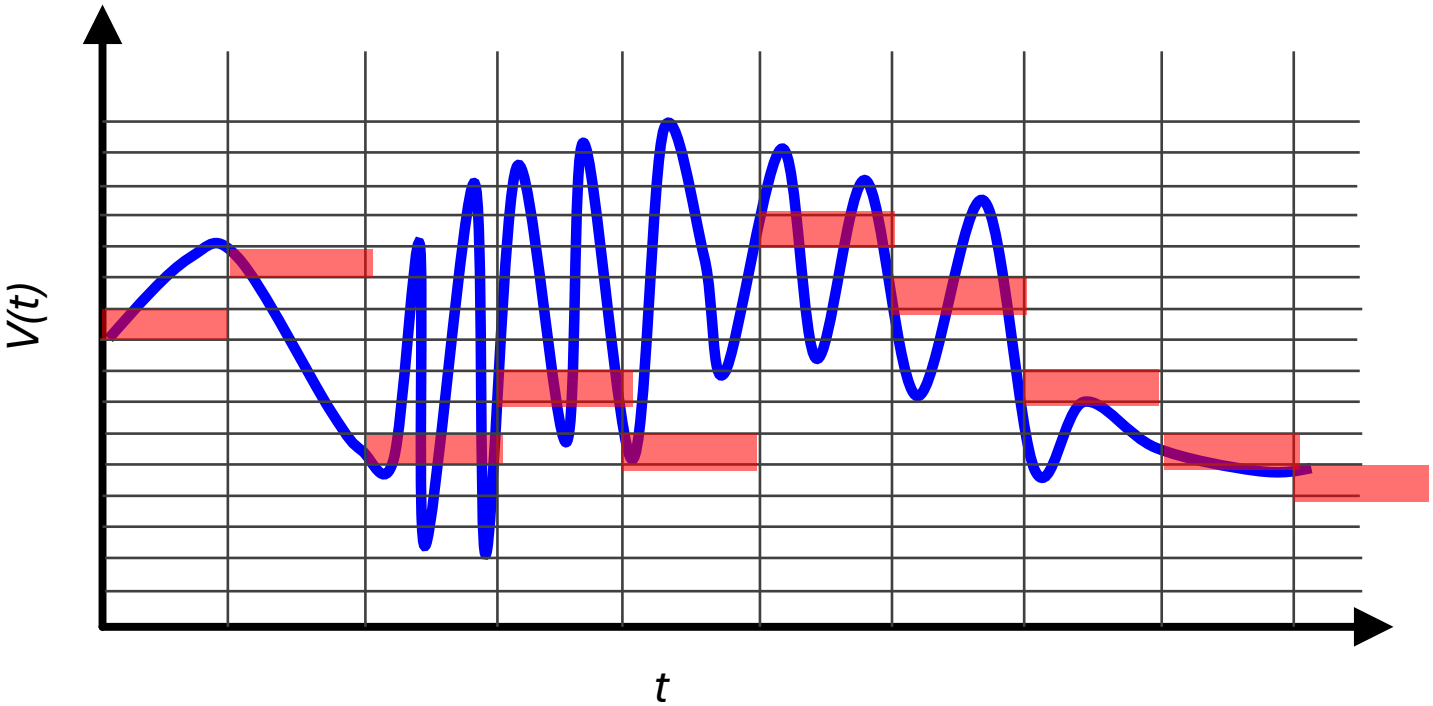


Let’s digitize it…at this sample rate we shouldn’t be able to capture it



4 bit value encoding

Discretization in Time and **Quantization** in Value



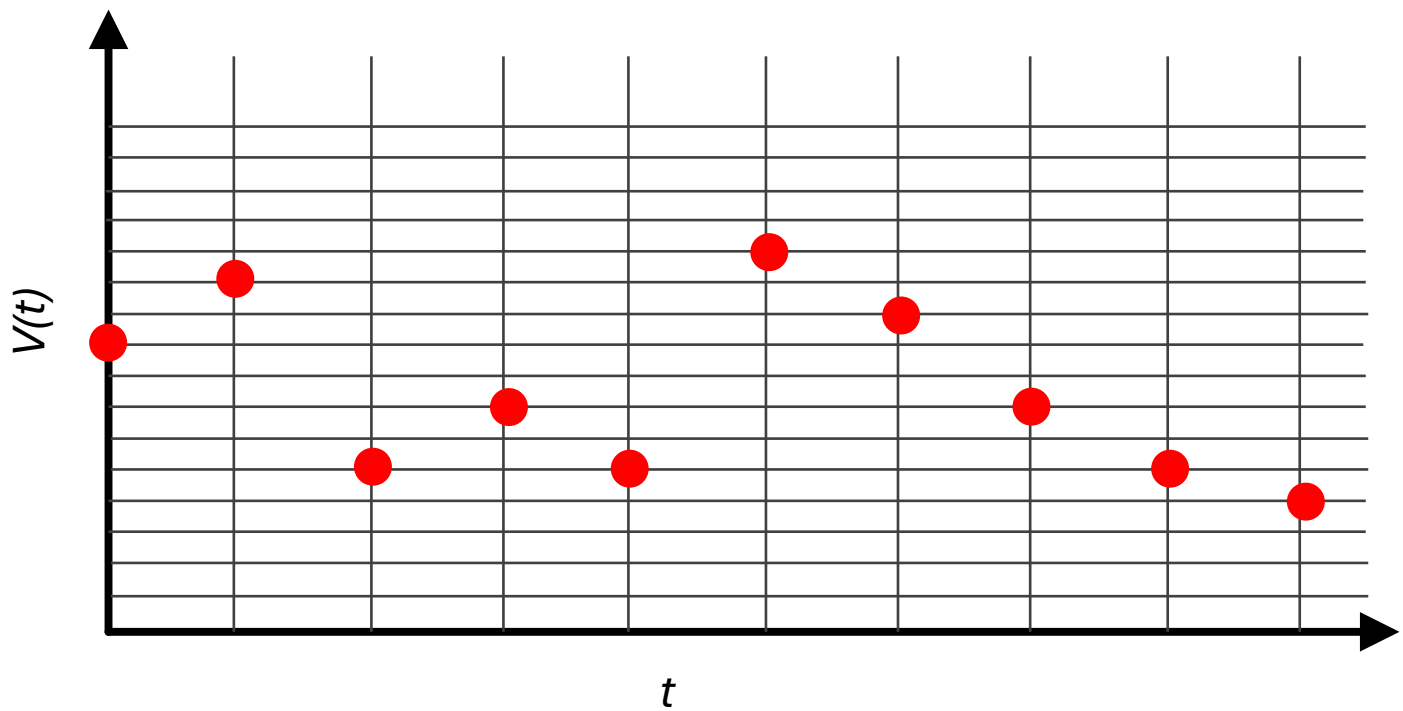
v[n] = [9,11,5,7,5,12,10,7,5,4,]

4 bit value encoding

Store in memory

- v[n] = [9,11,5,7,5,12,10,7,5,4,]
- 10 4-bit values: need 40 bits in memory!
- Easy-peasy one-two-threesy

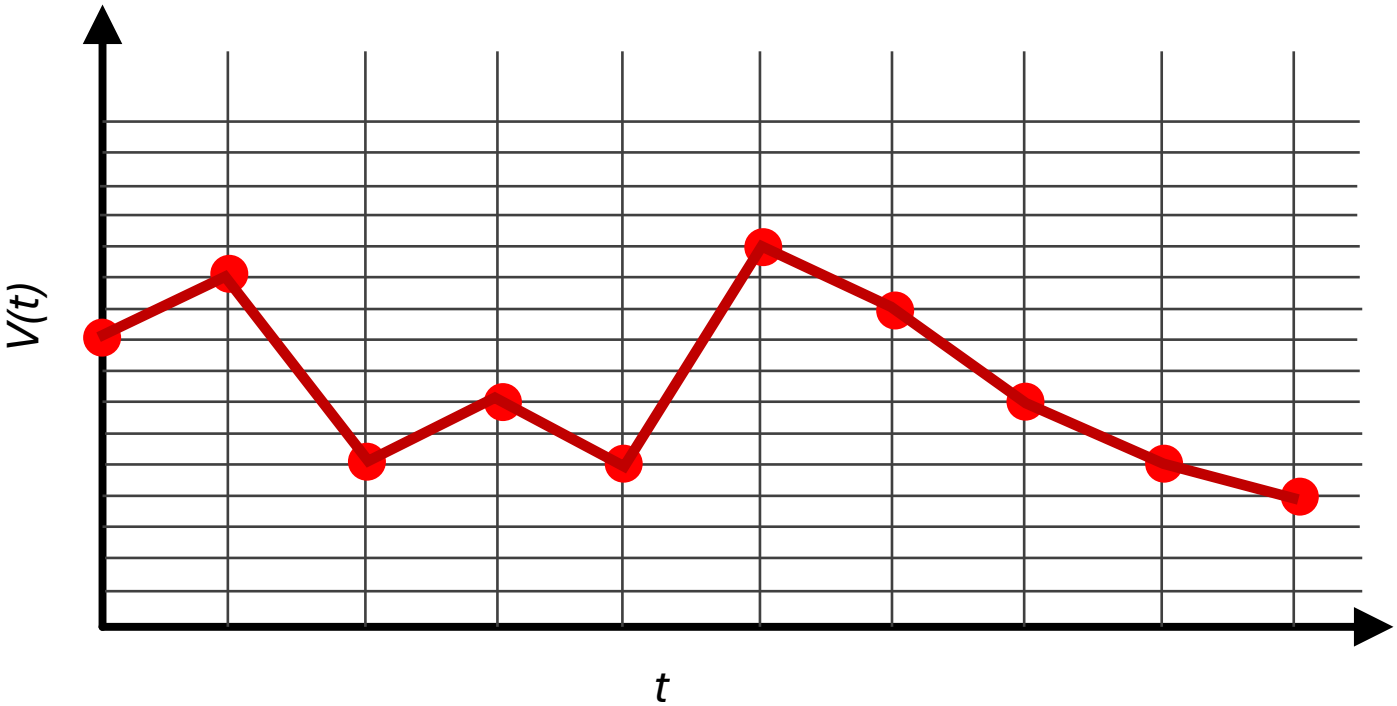
Reconstruct



v[n] = [9,11,5,7,5,12,10,7,5,4,]

4 bit value encoding

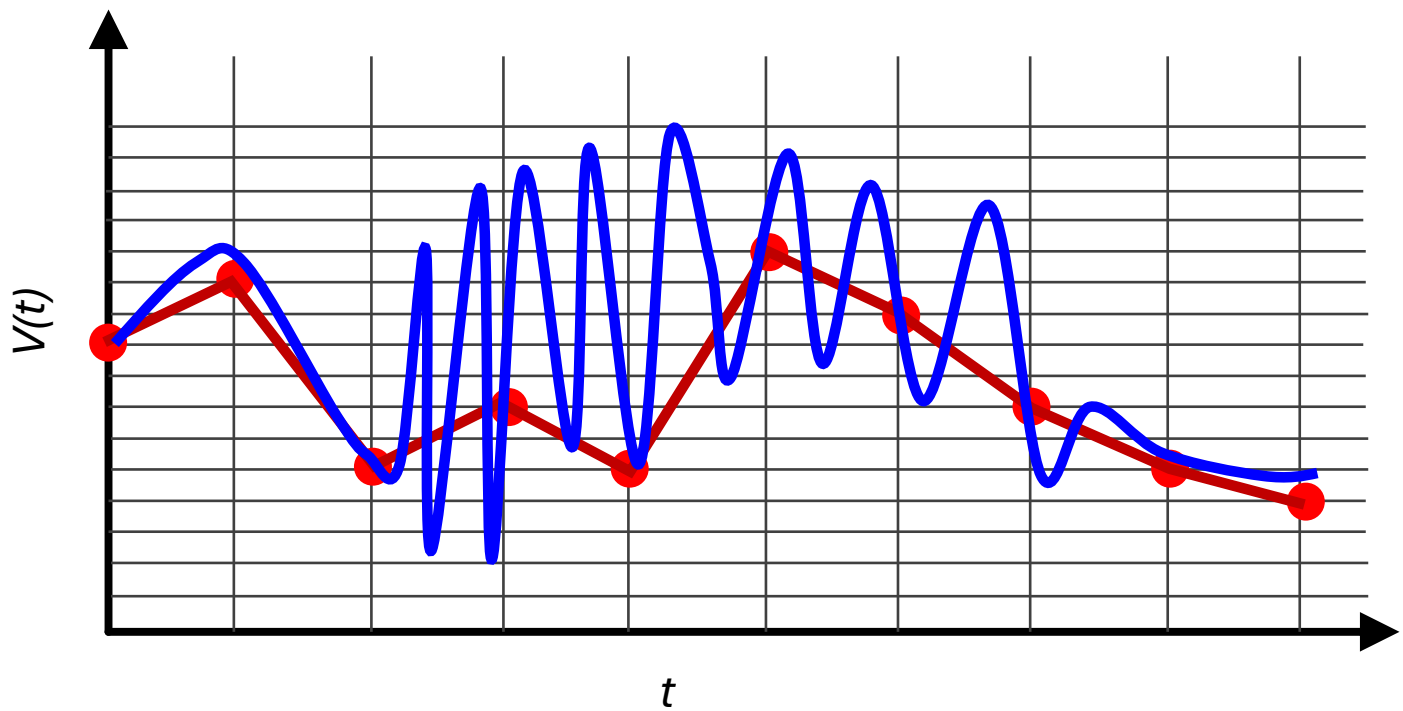
Reproduce



v[n] = [9,11,5,7,5,12,10,7,5,4,]

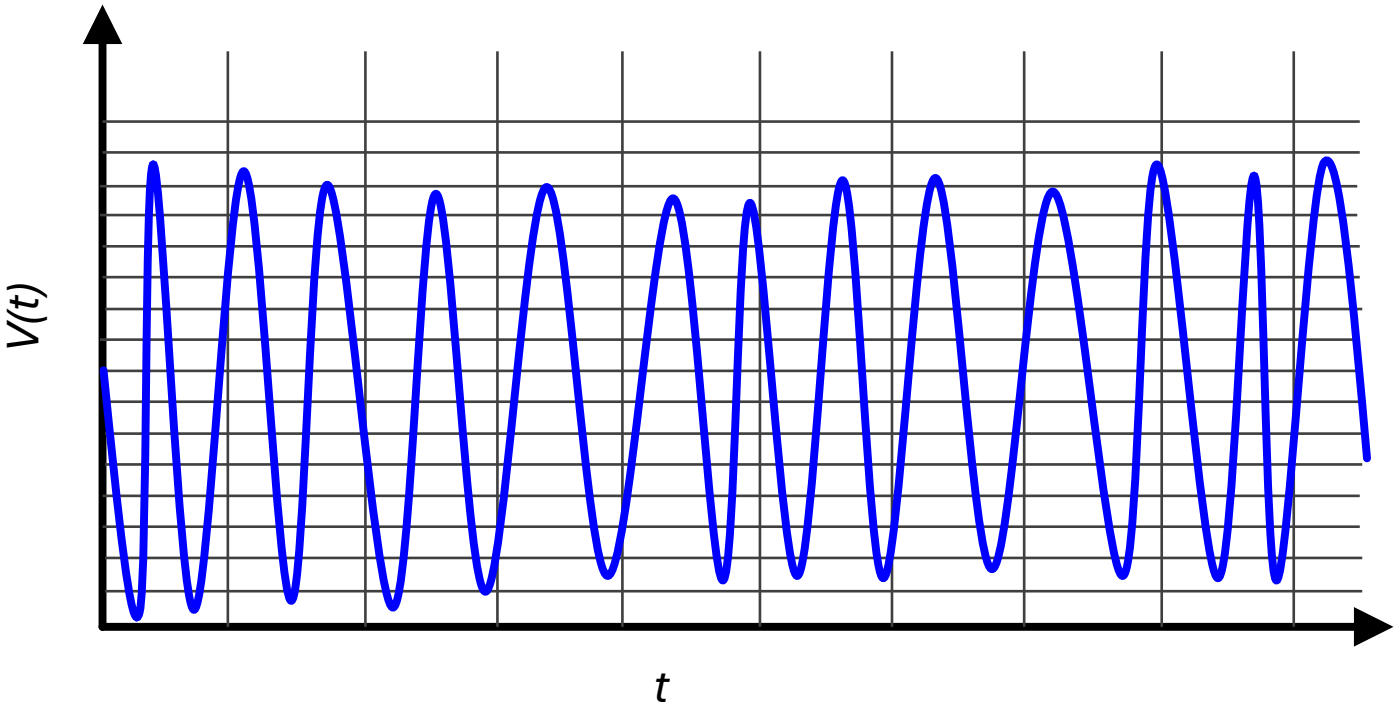
4 bit value encoding

Compare to original… Did not Capture the high-frequency Wiggles!

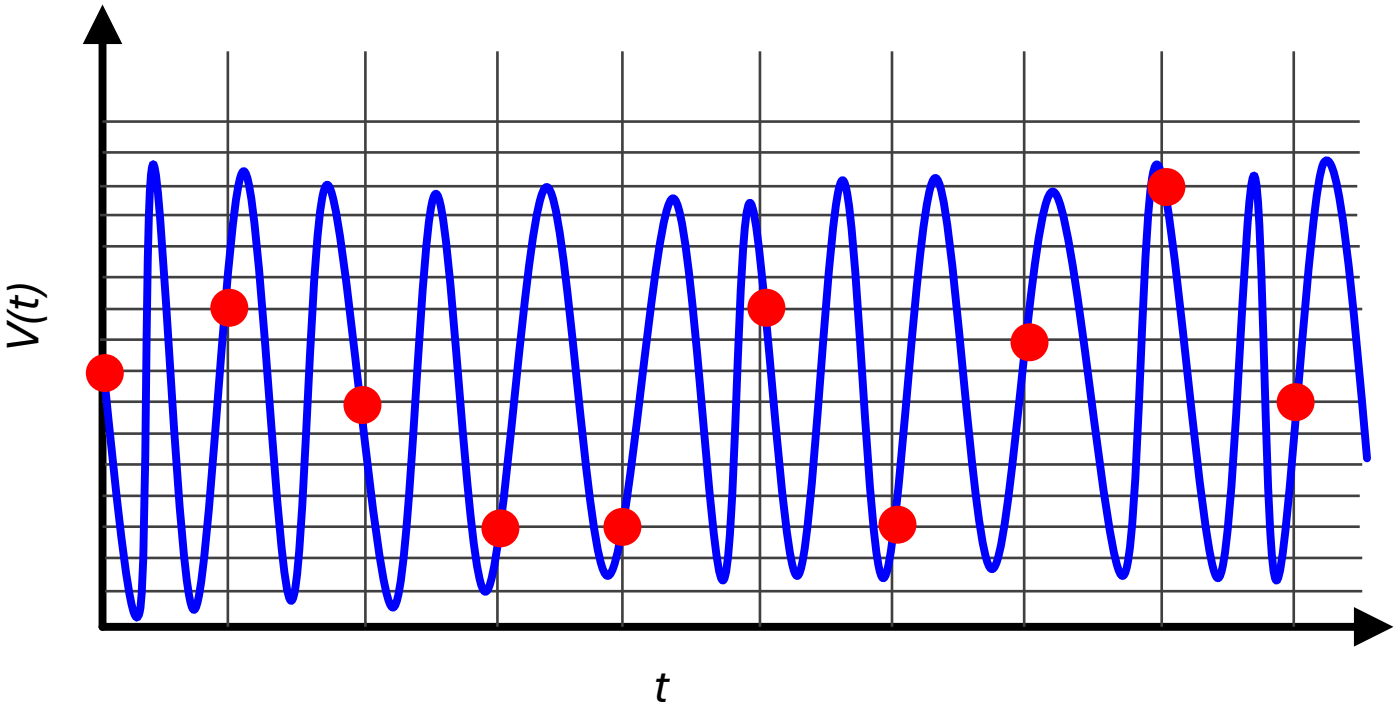


*Great….but we still captured something! What **<u>is</u>** that signal expressed by the red interpolation?*

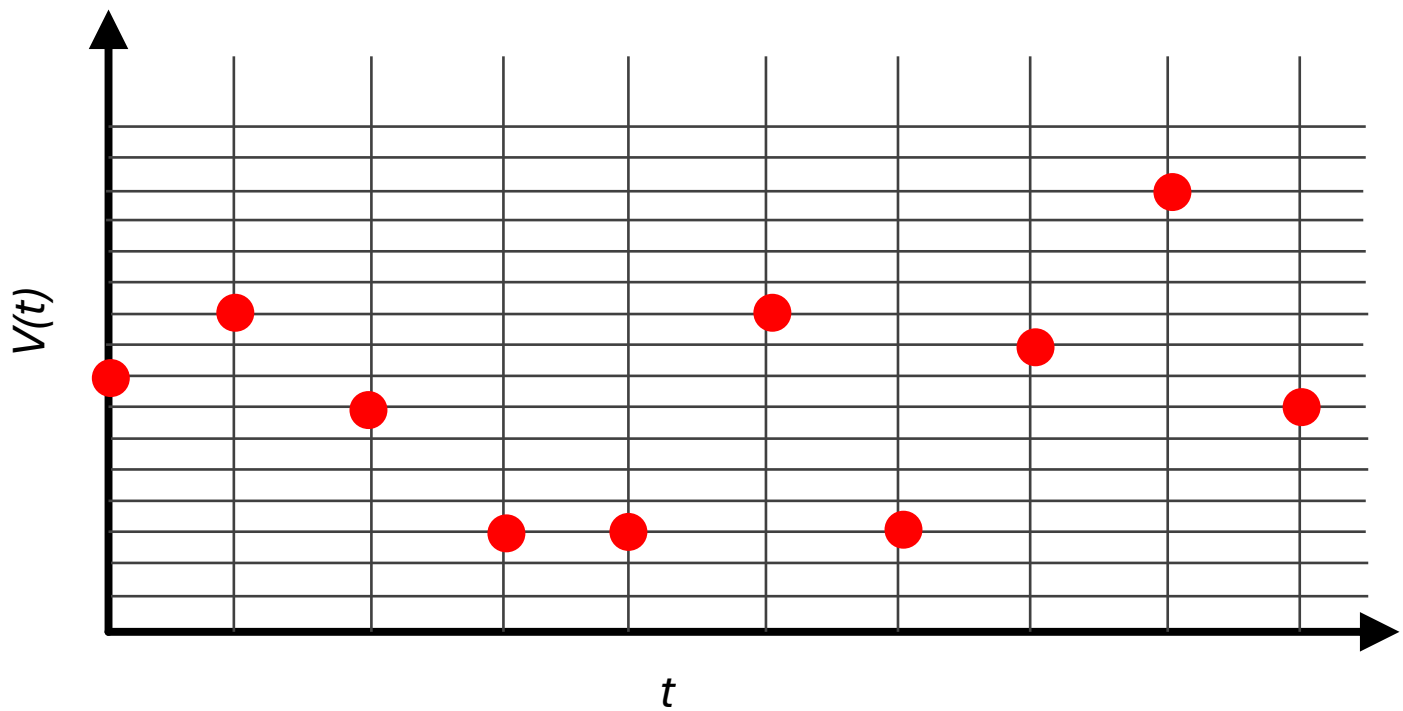
Consider this…



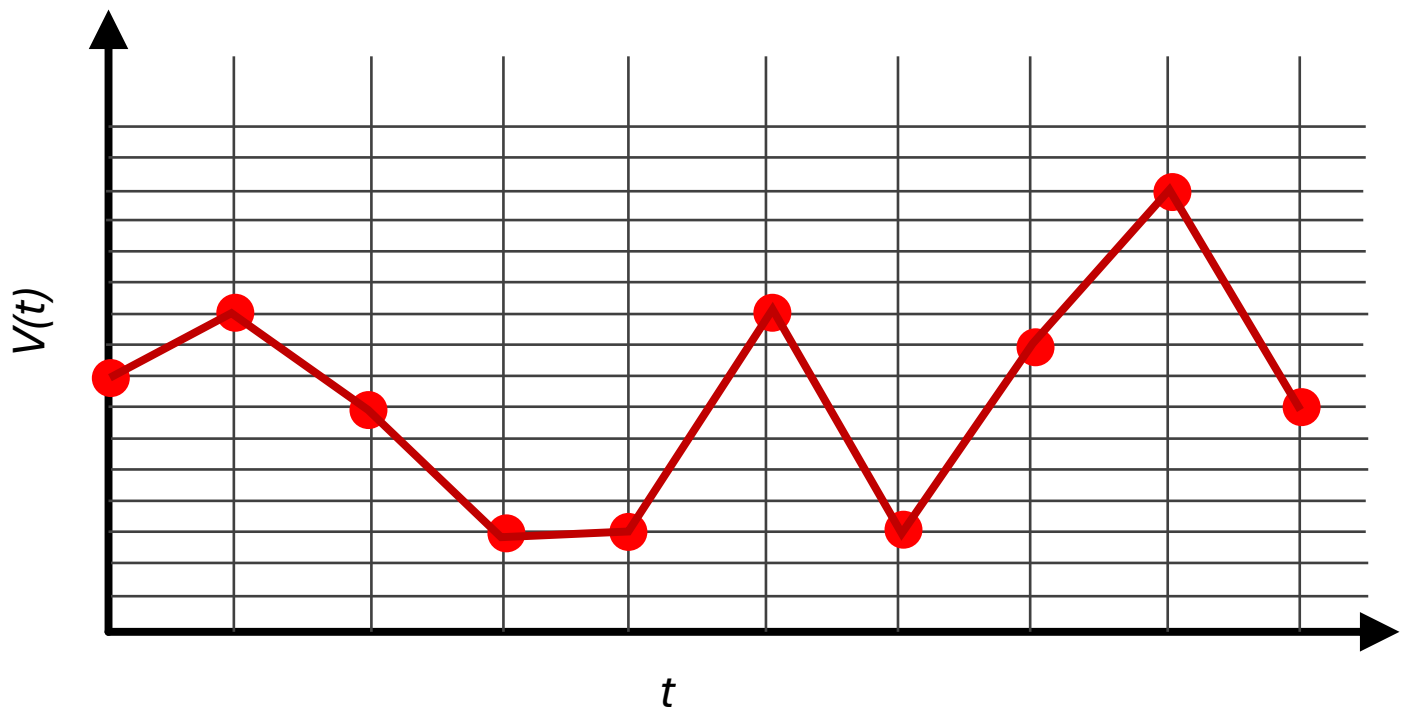
Sample it…



Store it…

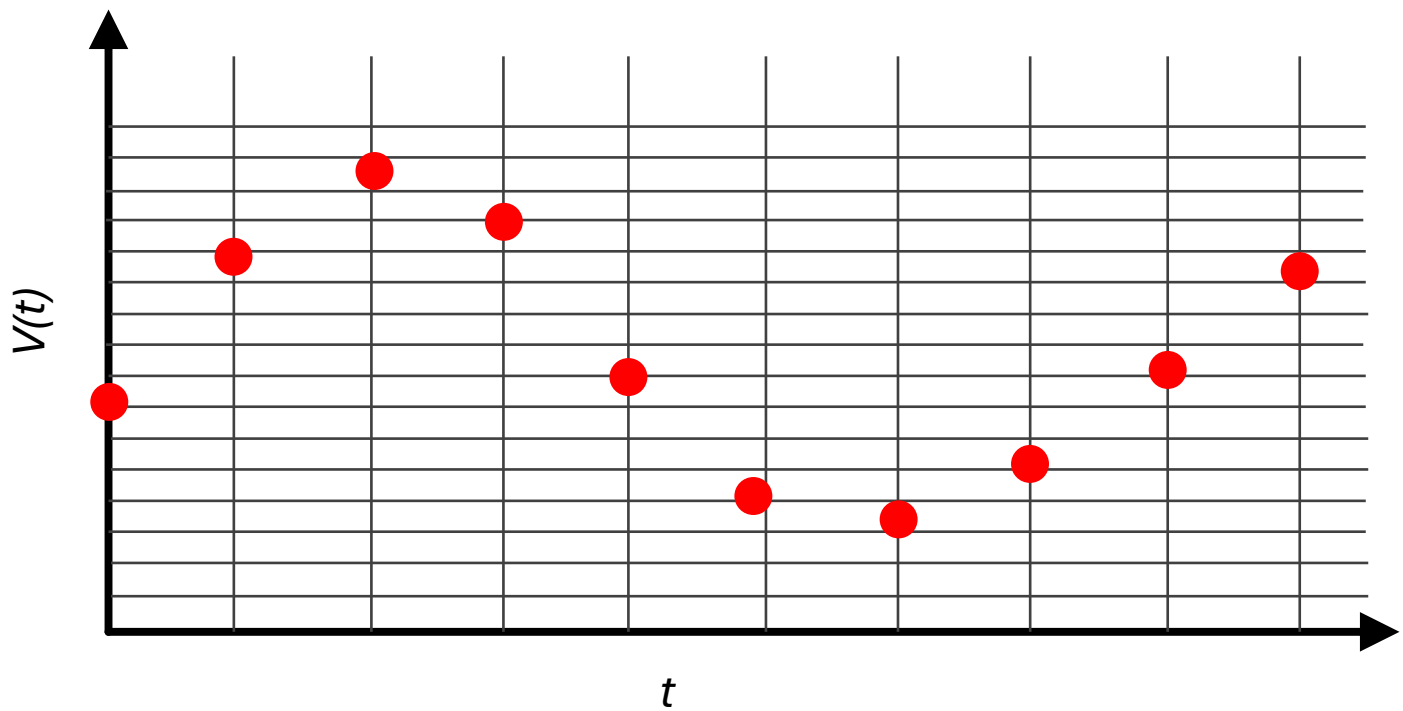


Reconstruct it…

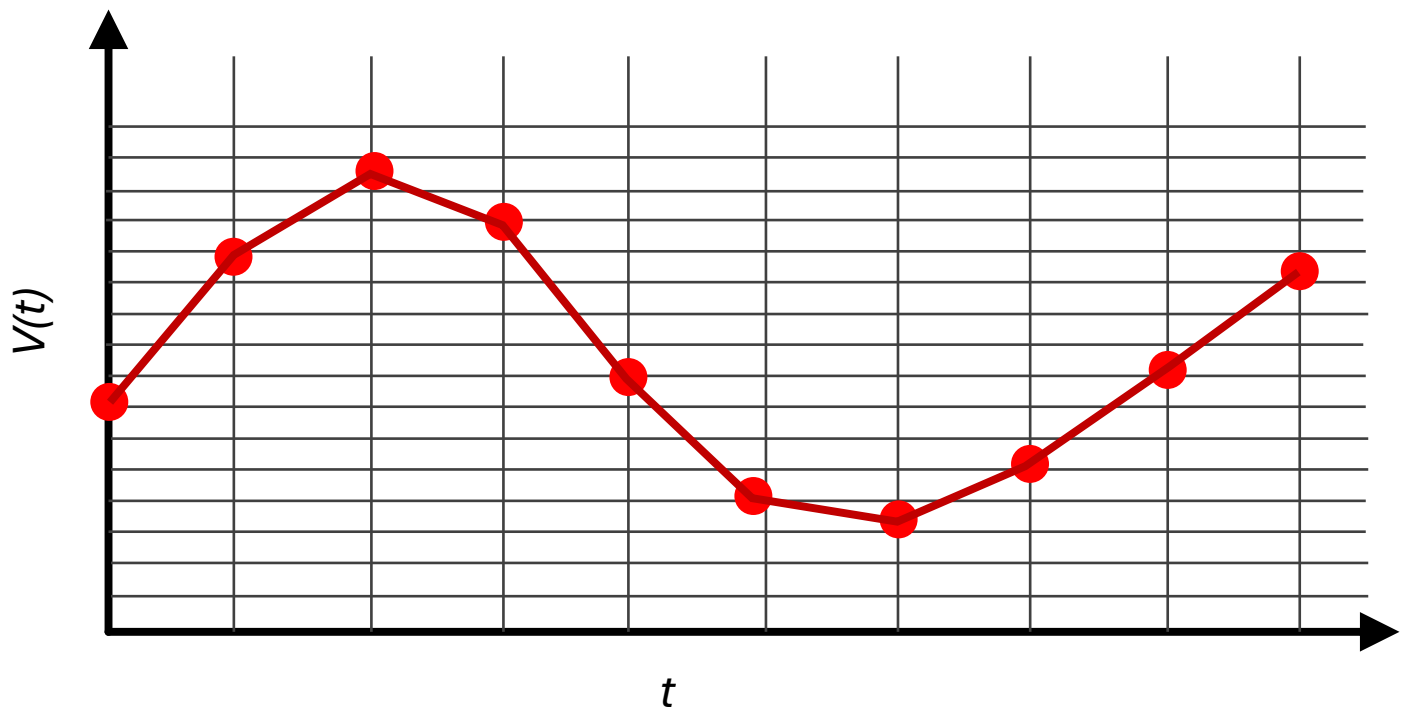


We’ve created a a different signal from what was before! WTH?

Or Consider this…
if we start with this data…

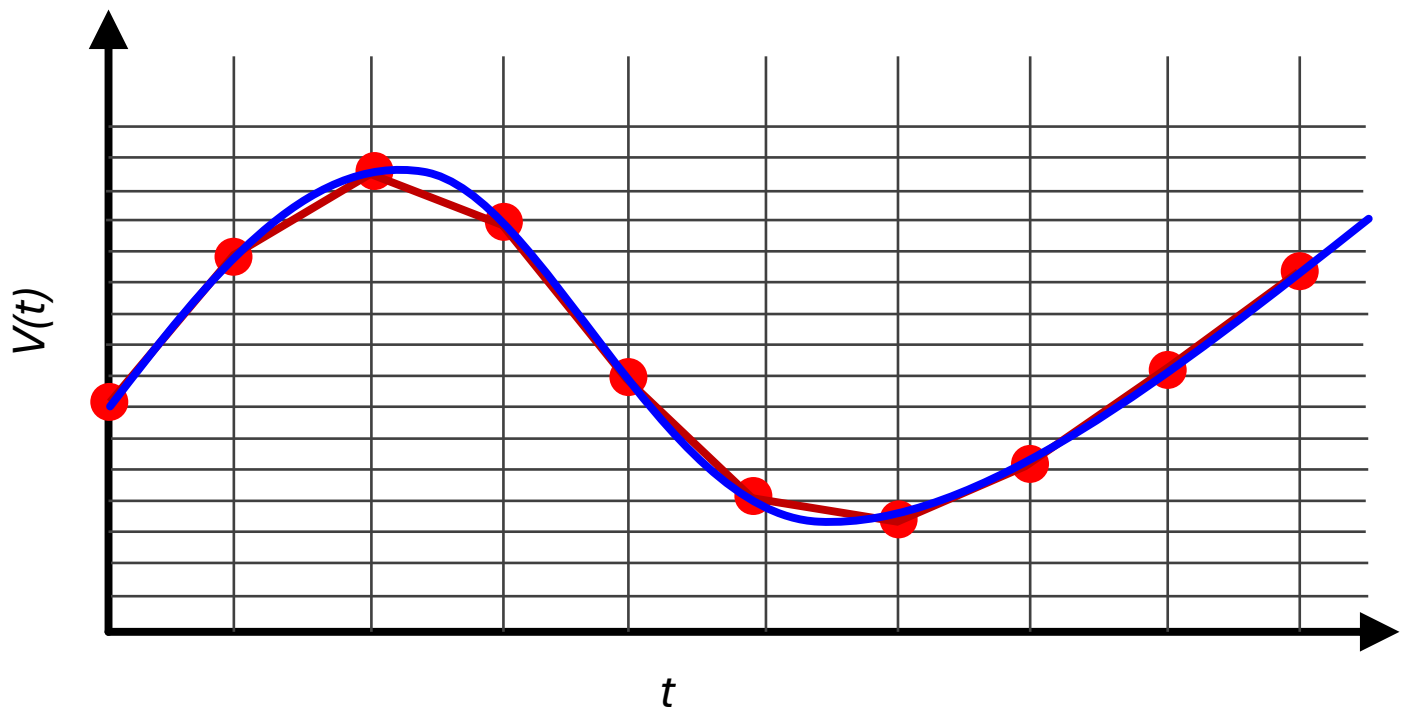


And we Reconstruct the signal…is this ok?

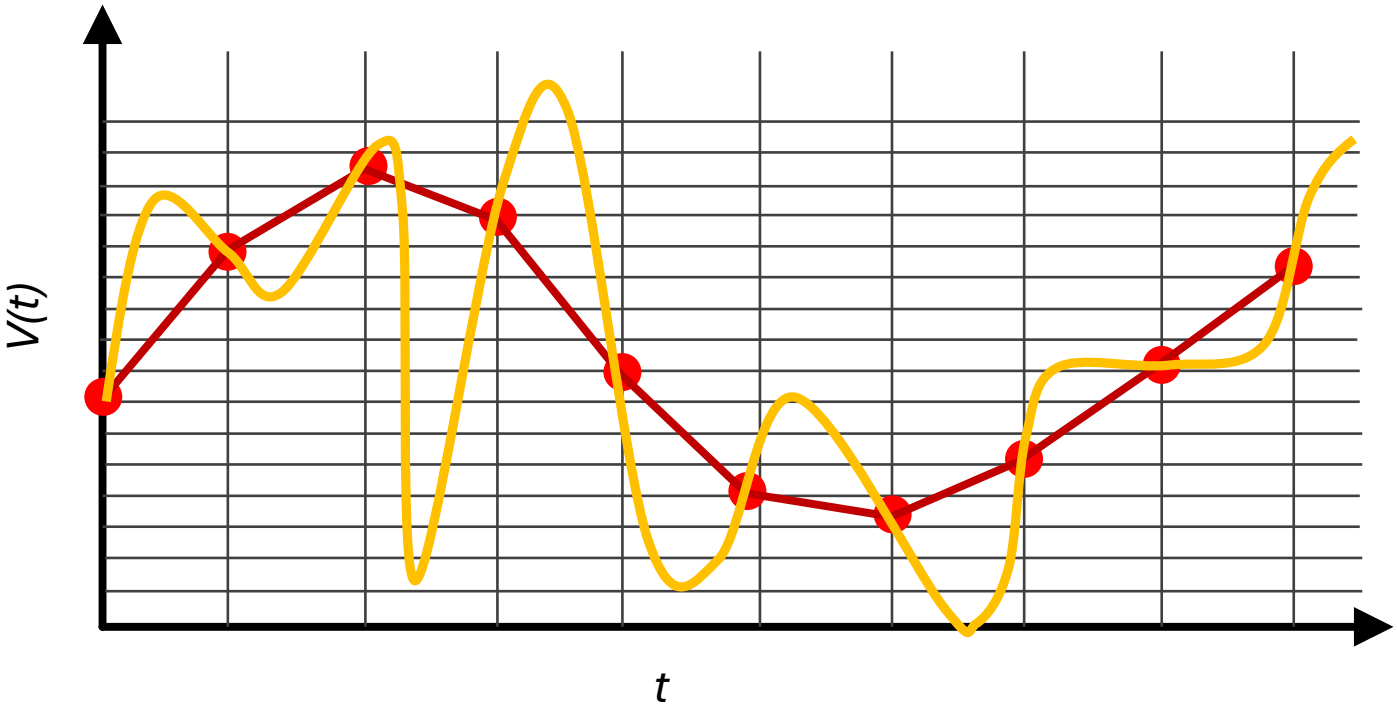


First-order hold (connect-the dots)

If it came from this, ok… but…

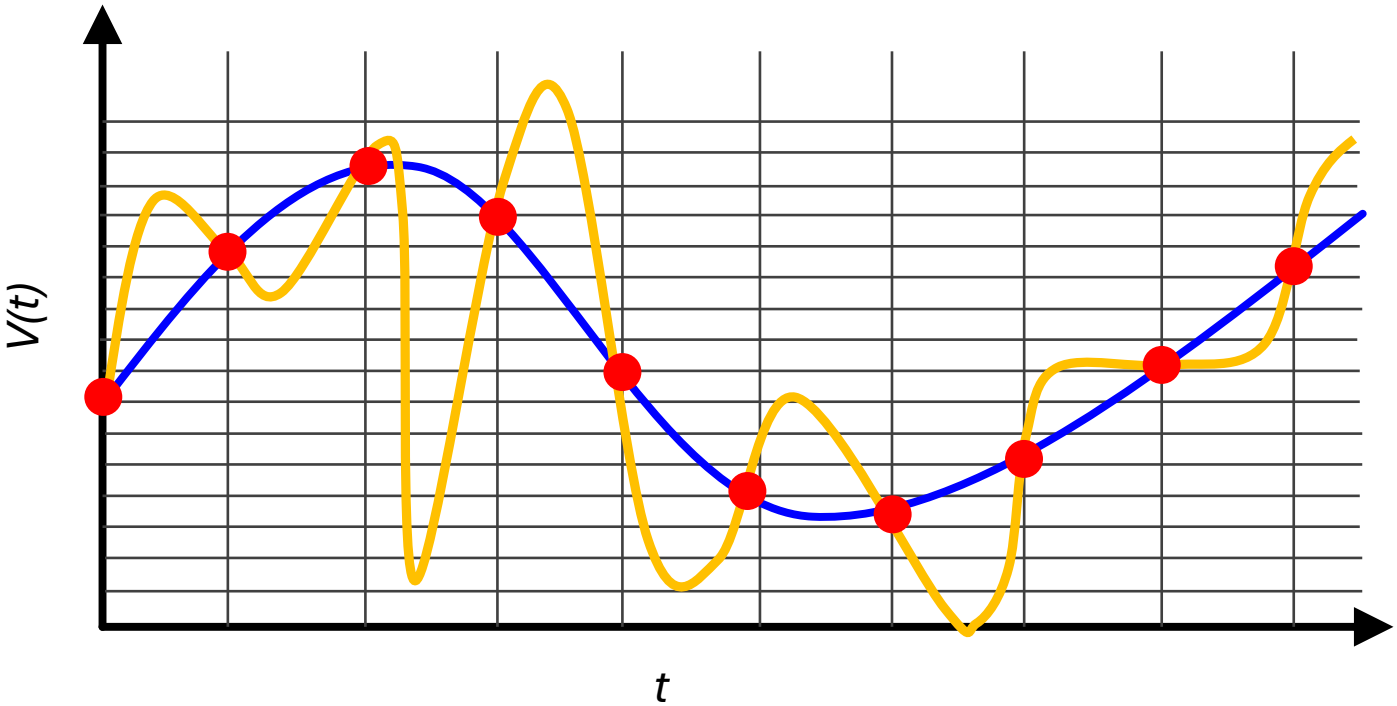


It could have also come from this…Uh oh



First-order hold (connect-the dots)

Which one Made the Signal



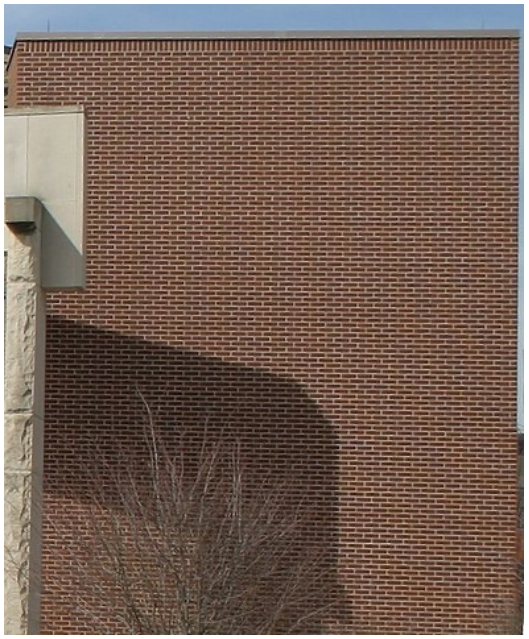
There's ambiguity in what those samples could represent…that means it really doesn't convey much, if any, information

Aliasing

- While we can't fully capture and reproduce signals with a frequency higher than the Nyquist sampling rate, it doesn't mean they **won't** have an impact!
- Energy from that high frequency will leak into the frame…a form of "spectral leakage"
- A sample rate of f_s can fully capture all information in a signal if and only if, the highest frequency in that signal is at or below $\frac{f_s}{2}$!
- **If you don't do this**, aliasing will appear (higher frequencies appear as a different signal (an "alias")) that can be expressed with the sample rate

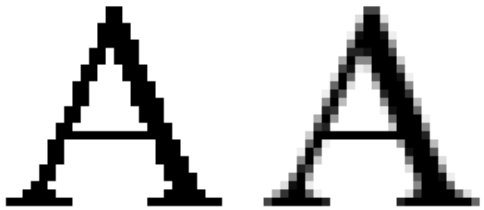
Aliasing Can Happen in Space too

- Just like there are temporal frequencies (in time), images have spatial frequencies.
- Same issues arise!



Anti-alias Filtered

Not Anti-alias Filtered

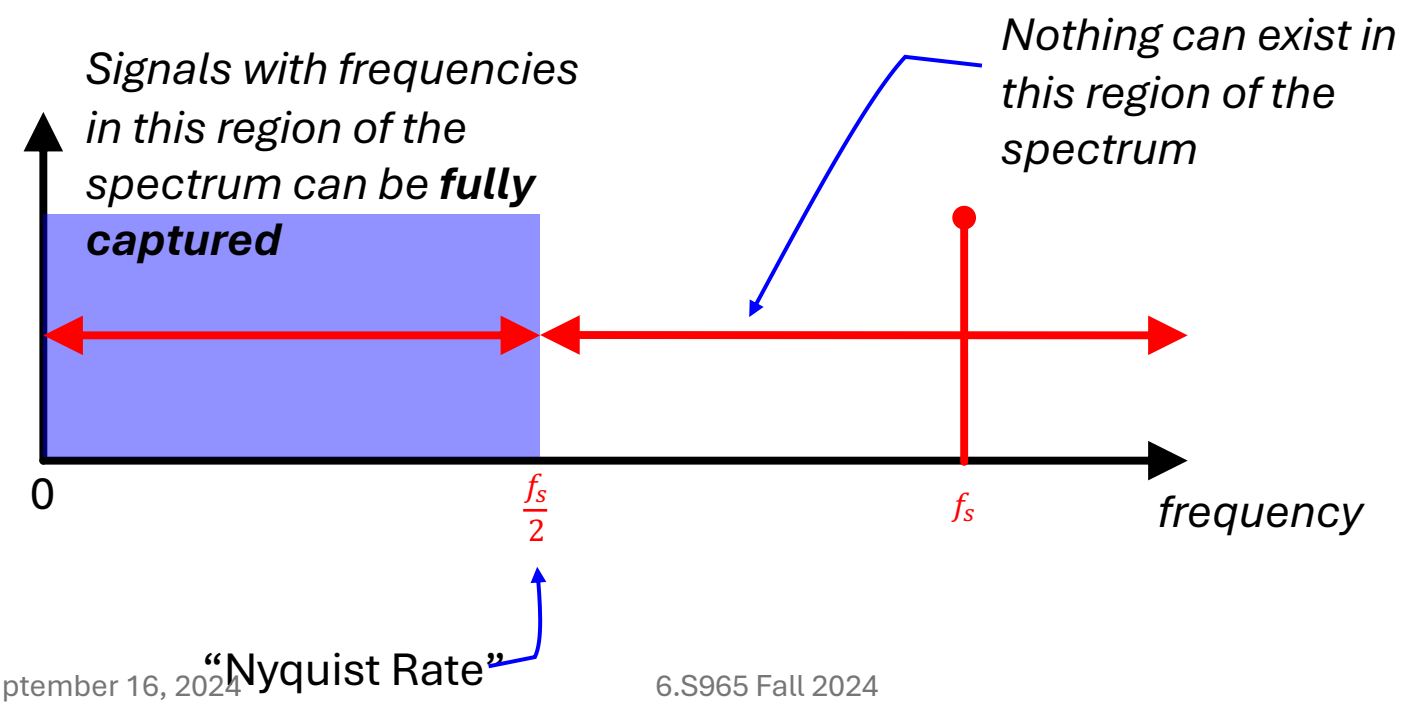


This font has been processed with an anti-alias filter to prevent artifacts when displayed

https://en.wikipedia.org/wiki/Aliasing

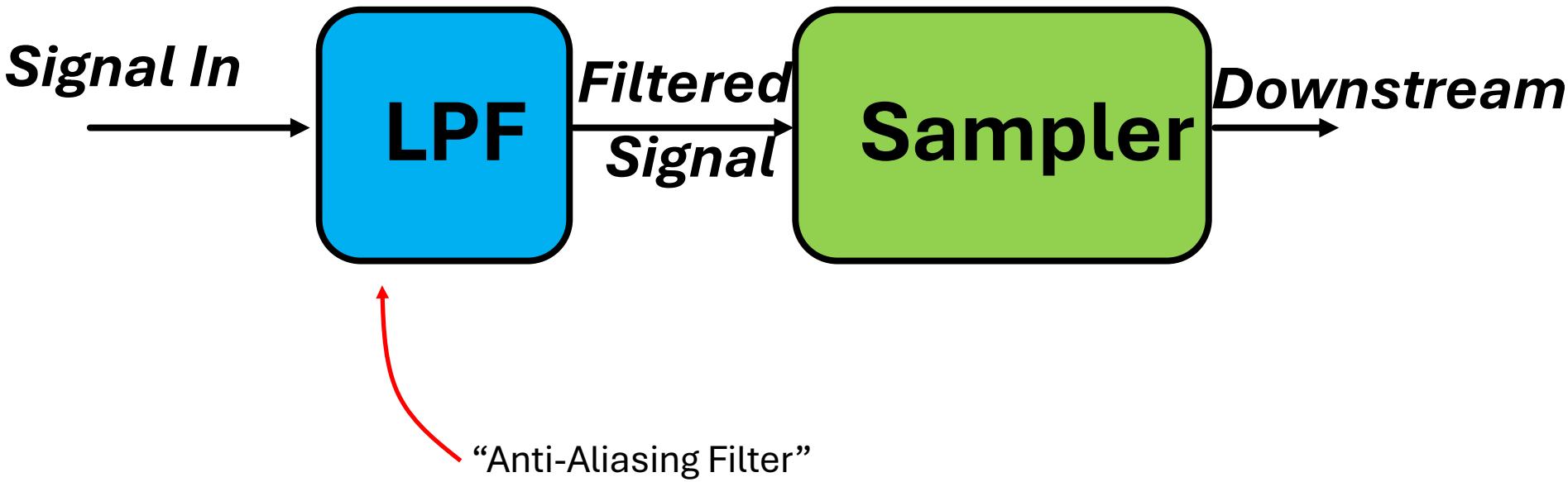
Solution

- The **ONLY** way to guarantee that a set of discrete points can unambiguously represent a signal is to guarantee that prior to sampling, we remove **all energy** that it exists in frequencies higher than the Nyquist Sampling Rate
- To do this we need a Low-Pass Filter!



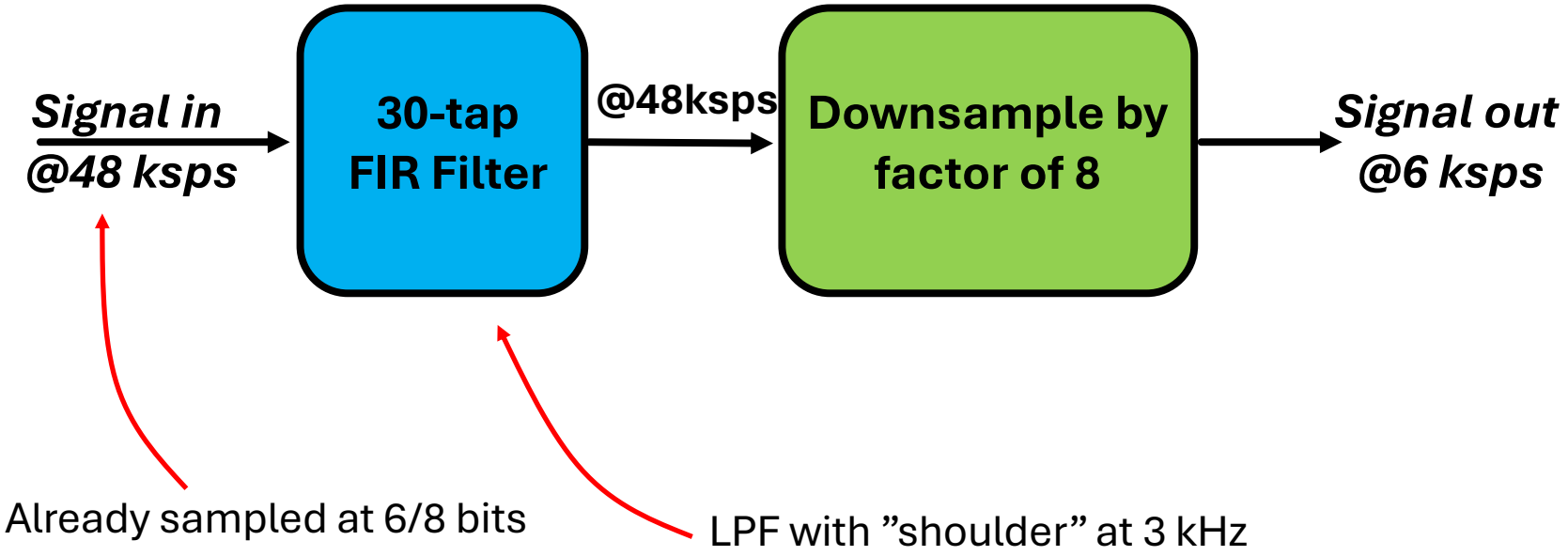
Low Pass Filter

- Prior to Sampling, we must be sure that our signal has no significant energy above our Nyquist Rate



Audio Sampling

- Since we're down-sampling by a factor of 8, to avoid aliasing (makes the recording sound "scratchy/metallic") we need to pass the incoming samples through a low-pass antialiasing filter to remove audio signal above 3kHz (Nyquist frequency of a 6kHz sample rate).



How Do You Actually Make a Filter?

- Several types of filters. Two big ones:
  - IIR: Infinite Impulse Response:
    - Uses past output history for filtering
  - FIR: Finite Impulse Response:
    - Uses input history for filtering
  - CIC: Cascaded Integrator Comb:
    - Special case of FIR mixed with down-samplers/decimators

Filters

- ***Stateful*** systems that analyze history signals to select for particular signal attributes:
  - **Low-pass Filter:** Lets through low-frequency signals
  - **High-pass Filter:** Lets through high-frequency signals
  - **Band-pass Filter:** Lets through selective group of frequencies
  - **Band-stop Filter:** Blocks selective group of frequencies
  - **Matched-Filter:** Values come from time-series of feature of interest being convolved with signal

Infinite Impulse Response Filter (IIR)

$$y[n] = \alpha \cdot y[n-1] + \beta \cdot x[n]$$

- The current output ($y[n]$) of the filter is based on the weighted sum of the previous output ($y[n-1]$) of the filter + the value of the input ($x[n]$)*
- Sometimes called a recursive filter: "y is based off of y is based off of y…"
- Information enters the system through x but its influence on the output is dependent on the values of α and β

*can also be based on multiple past values of y and x

Infinite Impulse Response (Modified)

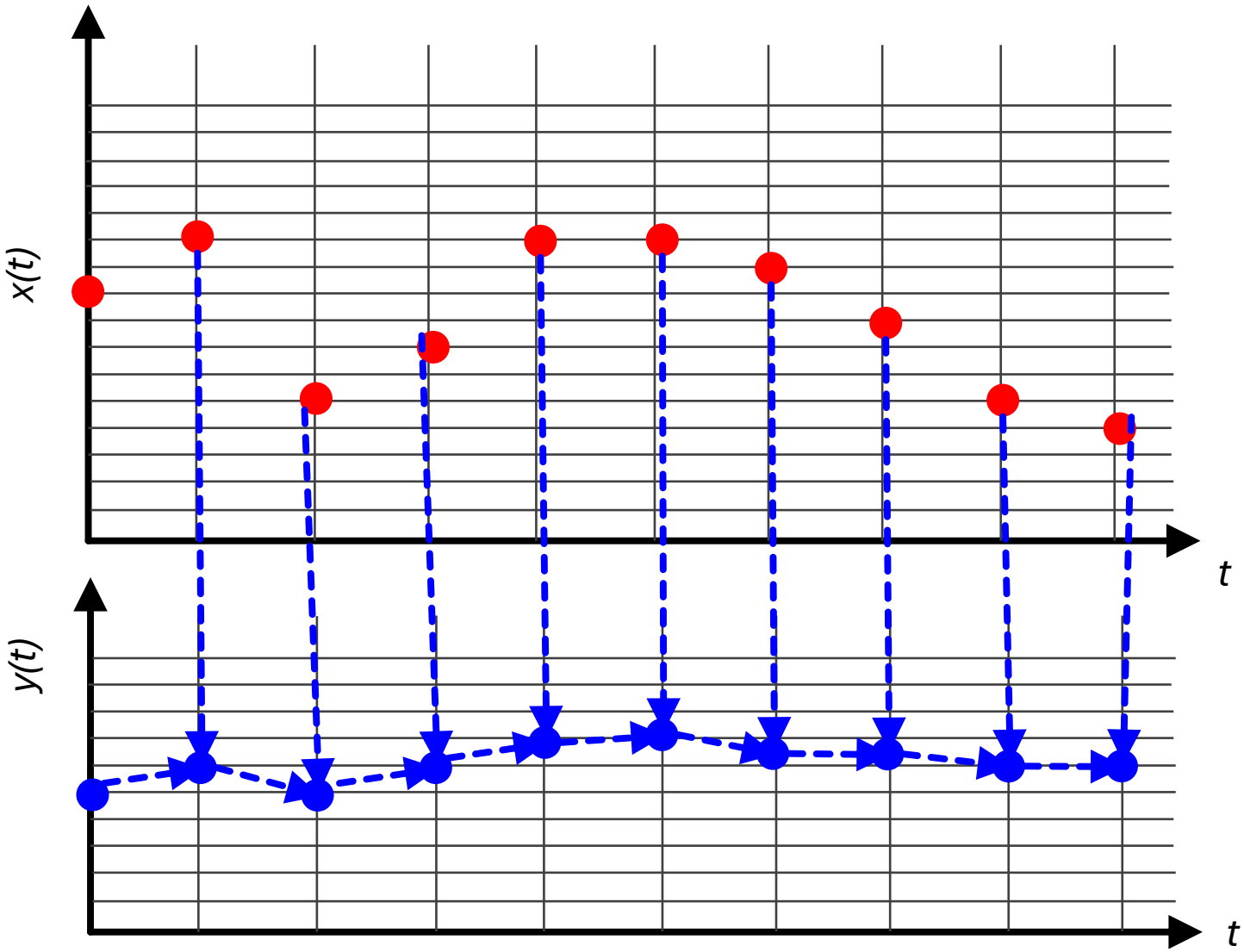
$$y[n] = \alpha \cdot y[n-1] + (1-\alpha) \cdot x[n]$$

$$0 \leq \alpha \leq 1$$

- Fix the relationship of the new input and old output to one variable α :
  - As $\alpha \rightarrow 1$ input has less weight (takes time for it to affect output…blocks more high frequency events)
  - As $\alpha \rightarrow 0$ input has more weight (output quickly follows input…allows through more high frequency events (and everything actually)

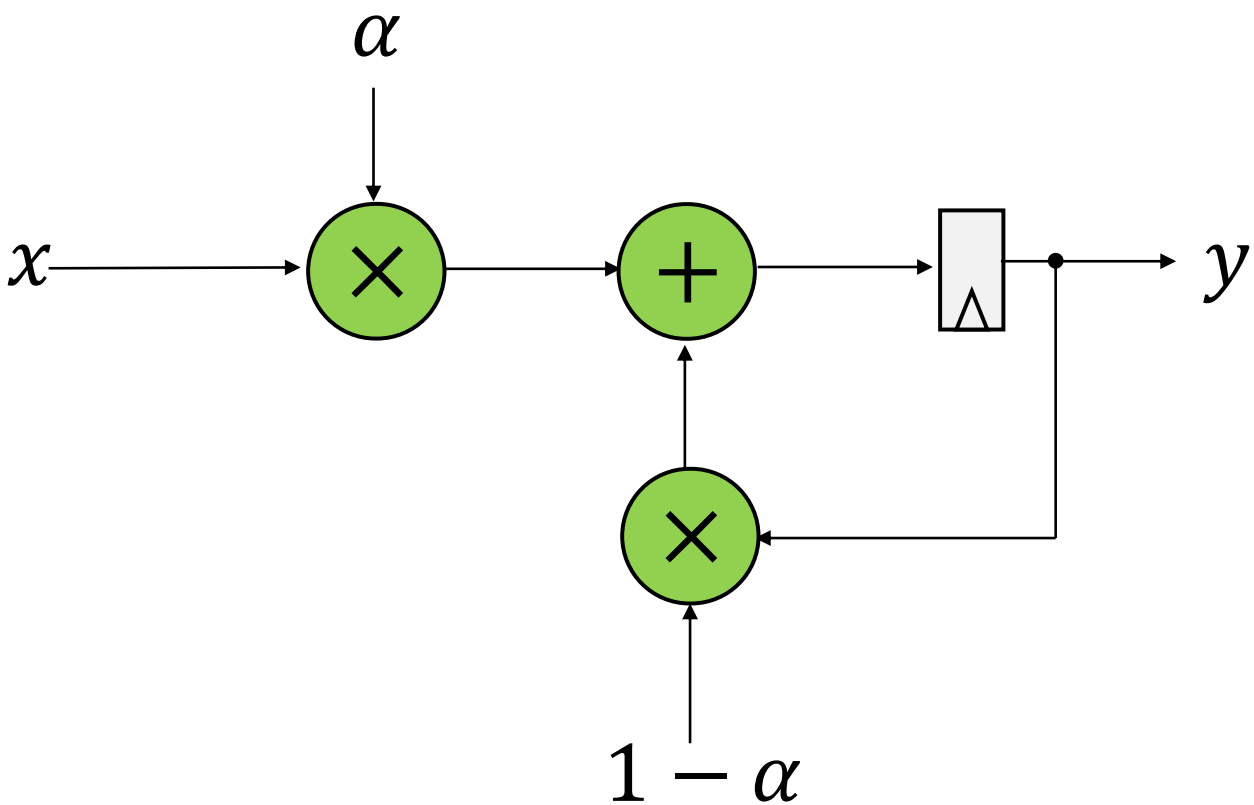
IIR Filter

$$y[n] = \alpha \cdot y[n-1] + (1-\alpha) \cdot x[n]$$



Infinite Impulse Response (Modified)

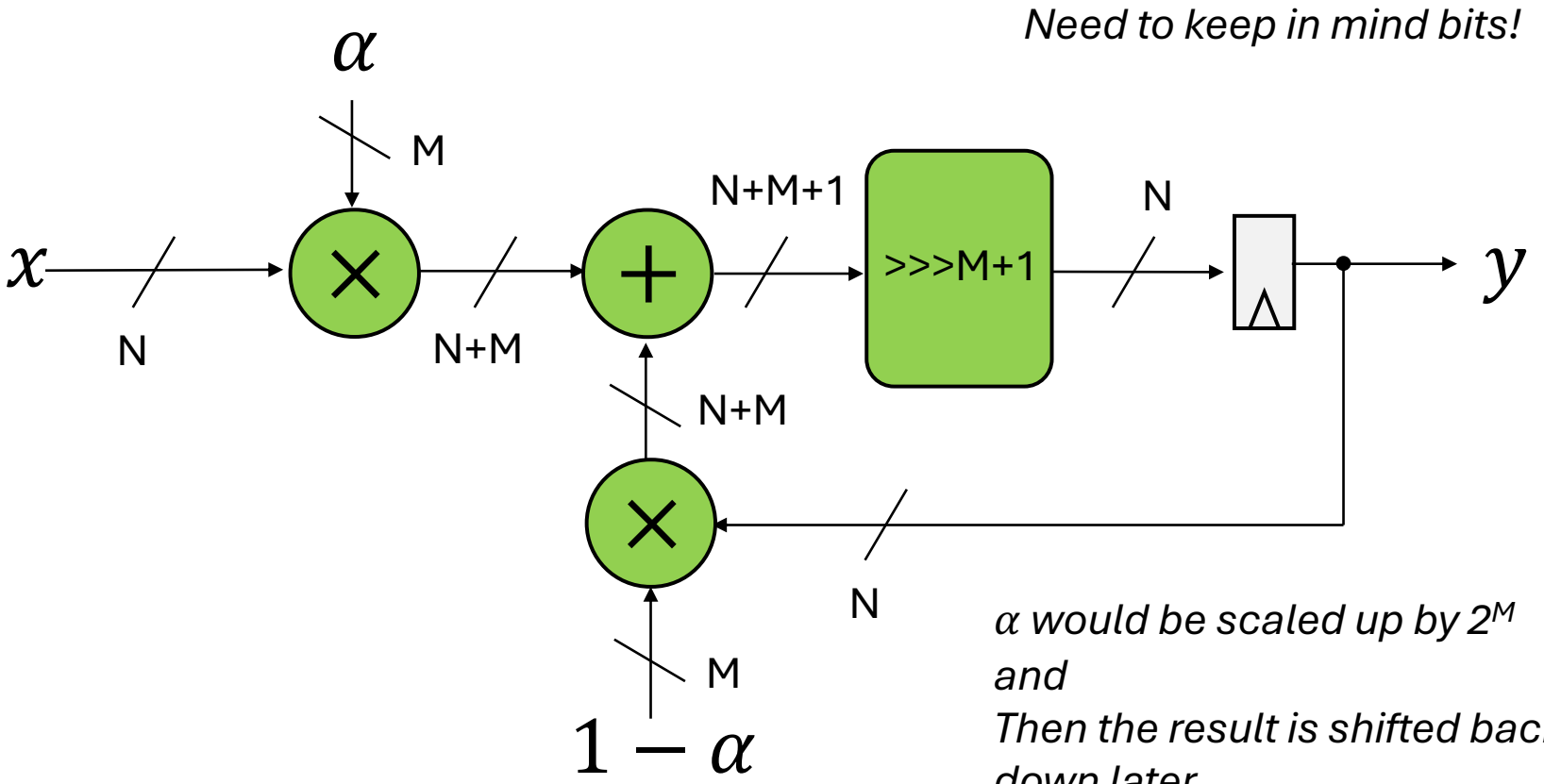
$$y[n] = \alpha \cdot y[n-1] + (1-\alpha) \cdot x[n] \qquad 0 \le \alpha \le 1$$



Infinite Impulse Response (Modified)

$$y[n] = \alpha \cdot y[n-1] + (1-\alpha) \cdot x[n] \quad 0 \le \alpha \le 1$$

Need to keep in mind bits!



α would be scaled up by 2^M and
Then the result is shifted back down later

IIR

- Computationally lightweight
- No very flexible, often poor performance since not a lot of parameters to adjust.

Finite Impulse Response

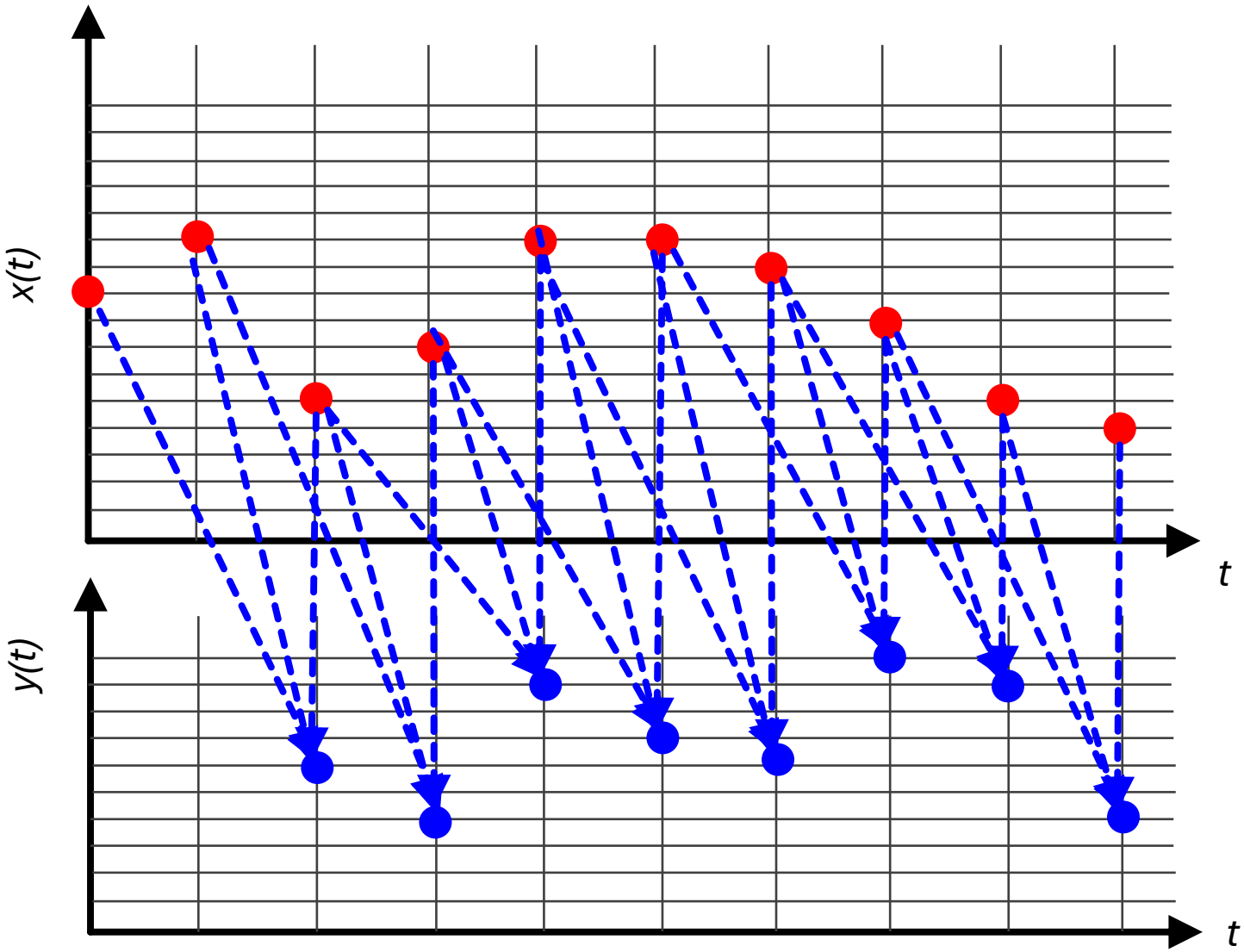
- Have the output be based off of a sliding window of the past history of the input.
- Literally just convolution basically

$$y[n] = b_0 \cdot x[n] + b_1 \cdot x[n-1] + b_2 \cdot x[n-2]$$

- Very powerful!! Huge flexibility in choosing those coefficients and can get a ton of behaviors!

FIR Filter

$$y[n] = b_0 \cdot x[n] + b_1 \cdot x[n-1] + b_2 \cdot x[n-2]$$



FIR Filters

- Extremely flexible
- Often times **many, many** "taps" long (N in 1000s is not uncommon)

$$y[n] = \sum_{k=0}^{N-1} b_k \cdot x[n-k]$$

- The values you pick for these taps are arrived at using a number of DSP-oriented algorithms (beyond scope of course…but in 6.341, etc)

FIR Filters

$$y[n] = \sum_{k=0}^{N-1} b_k \cdot x[n-k]$$

- Some online tools, Matlab, Python, Vivado all have tools that allow you to:
  - specify how you want your filter to look
  - Provide you the coefficients needed to generate that filter
- The b coefficients are generally provided as real numbers between 0 and 1. But since we don't want to do floating point arithmetic, we usually scale them by some power of two and then round to integers.
  - Since coefficients are scaled by 2^M, we'll have to re-scale the answer by dividing by 2^M. But this is easy – just get rid of the bottom M bits!
- More taps generally means you can get better response:
  - Closer to ideal filter!

FIR Filters

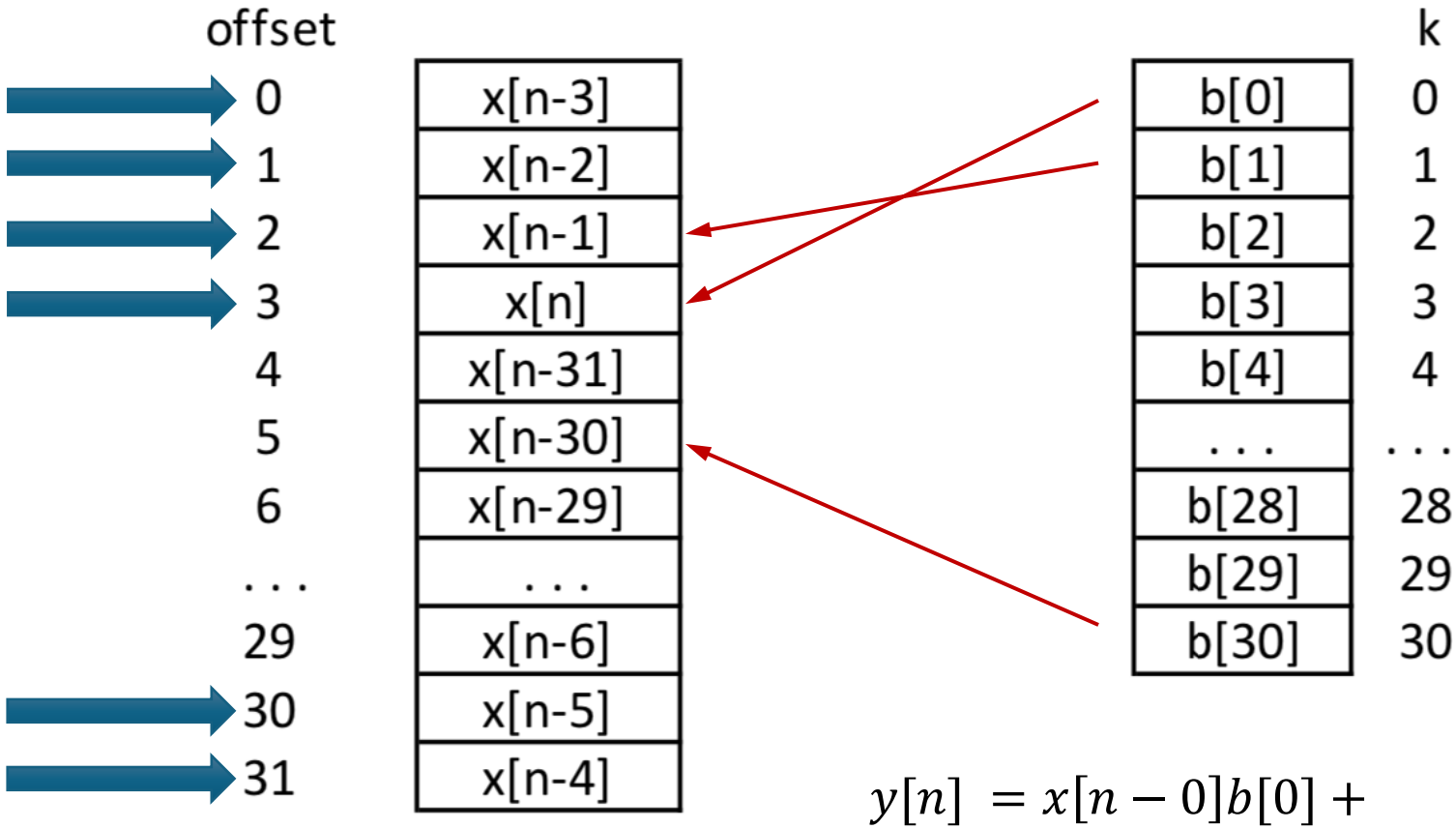
- They implement convolution, so can be much more than just “filters”
- You can use them to:
  - Remove complicated features to signals
  - Add complicated features to signals
  - Making an FIR filter “dynamic” can lead to systems that dynamically tune themselves.
  - Make a ”matched filter” to look for features.
- Very much a work-horse type module.

FIR Filter (Iterative Design)

$$y[n] = \sum_{k=0}^{N-1} b_k \cdot x[n-k]$$

- For audio and mid-frequency phenomena, usually plenty of clock cycles exist between each audio cycle anyways (you have 2000 clock cycles of 100 MHz between each audio sample of 48 ksps audio!)
- Just make a low-resource state-machine-based module.
- After every sample, do each multiply-accumulate for each tap. As long as you have enough cycles, you can do thousands of taps. Can even break up into more

Circular Buffer/Pointer in Action

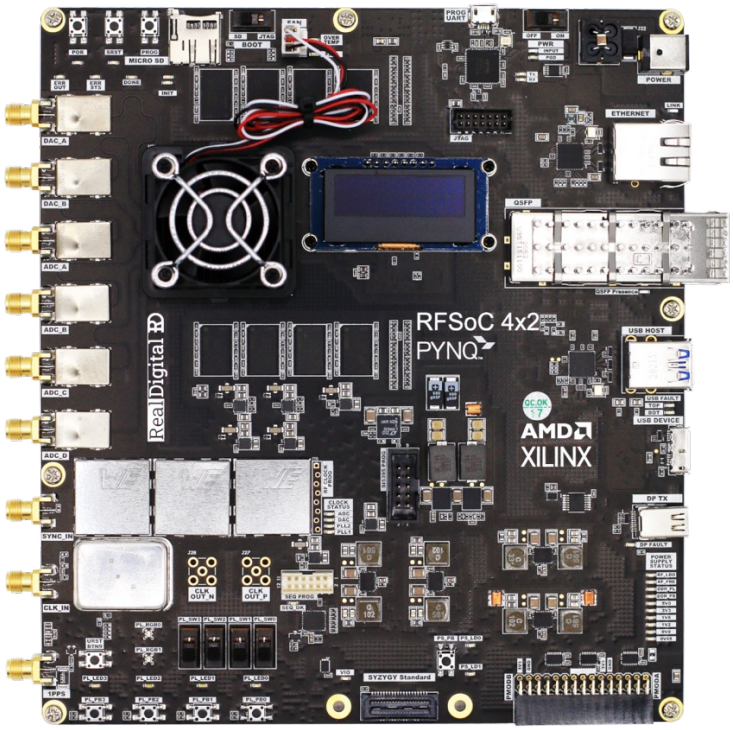


$$y[n] = \sum_{k=0}^{30} x[n-k]b[k]$$

$$y[n] = x[n-0]b[0] + x[n-1]b[1] + \ldots x[n-30]b[30] +$$

6.S965 RFSoC

- UltraScale+ ZU48DR:
  - 38 Mb of BRAM
  - +22Mb of UltraRAM
  - 4272 DSP slices
  - 930,000 Logic Cells
  - **Four 5-Gsps 14 bit ADCs**
  - **Two 10-Gsps 14 bit DACs**
  - Four 1.3 GHz ARM 53 processors
  - Two Real-time 533 MHz ARM processors
- Board has 4GB of DDR4 for FPGA portion (”PL”) and 4 GB of DDR4 for processors (“PS”)



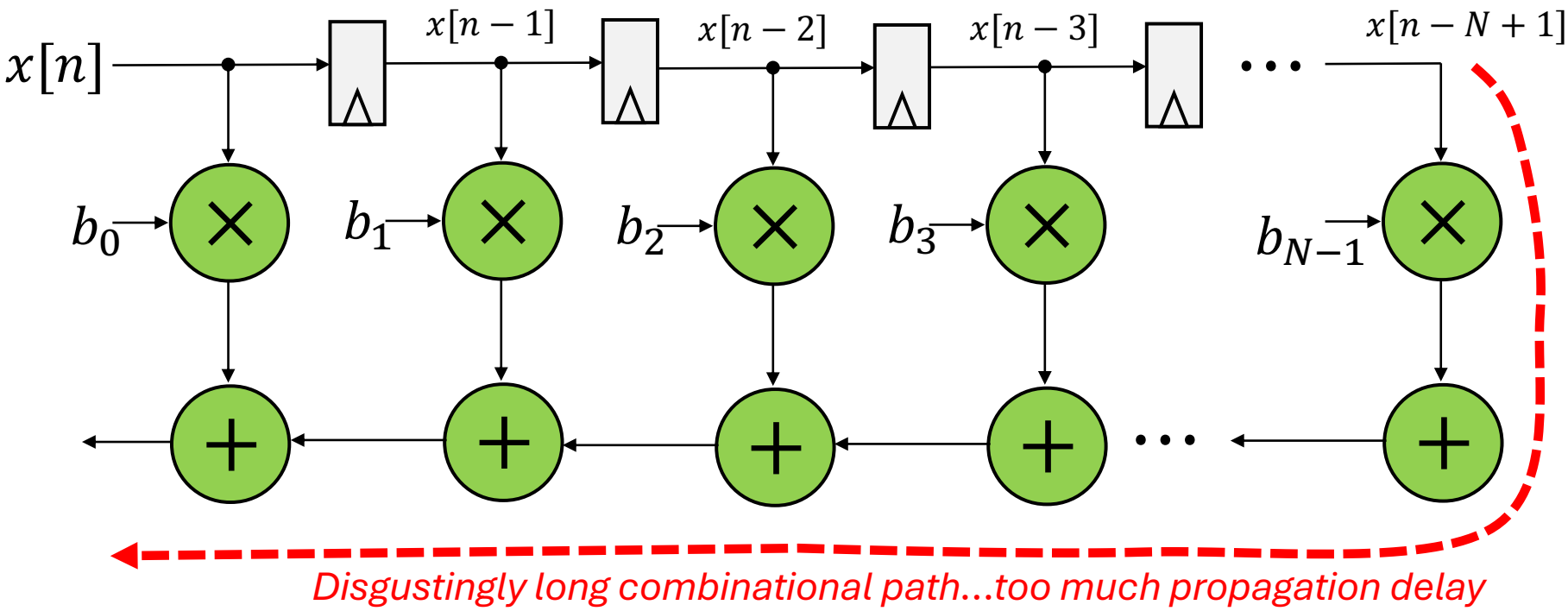
https://www.amd.com/en/products/adaptive-socs-and-fpgas/soc/zynq-ultrascale-plus-rfsoc.html#tabs-b3ecea84f1-item-e96607e53b-tab

How Much Data is That?

- The max the FPGA fabric can run is like 700 MHz or so.
- If ADCs run at 5 Gsps how many clock cycles

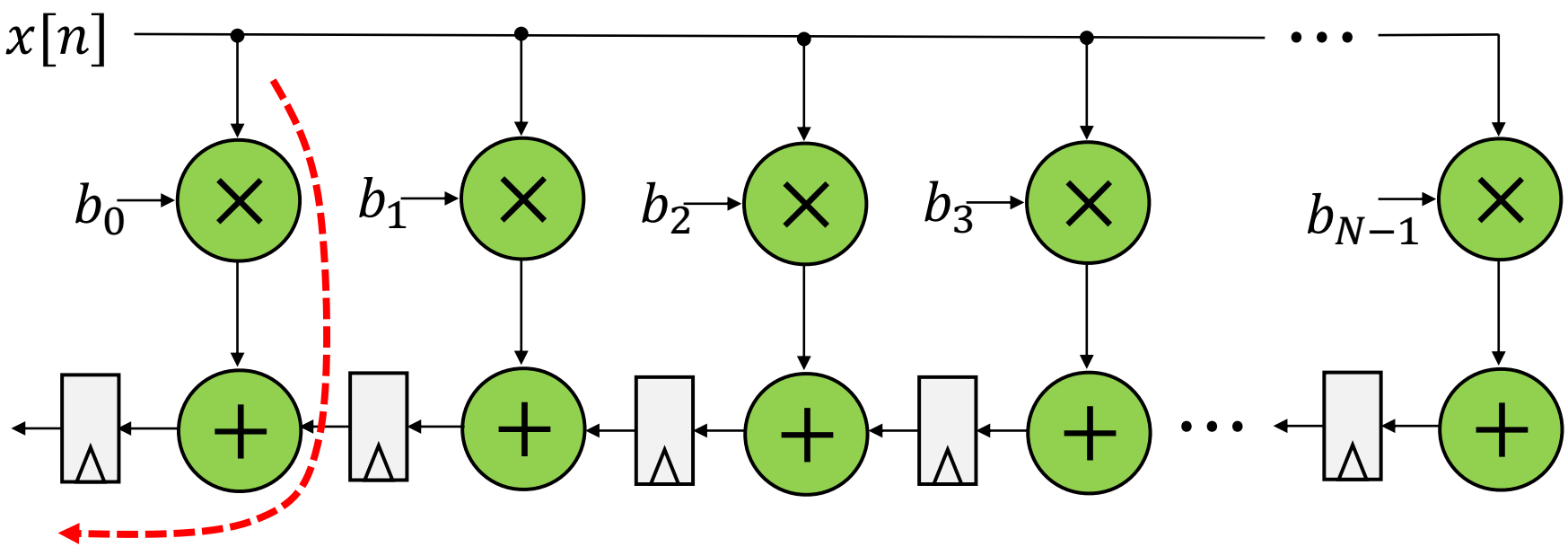
Finite Impulse Response

$$y[n] = \sum_{k=0}^{N-1} b_k \cdot x[n-k]$$



Finite Impulse Response (Modified)

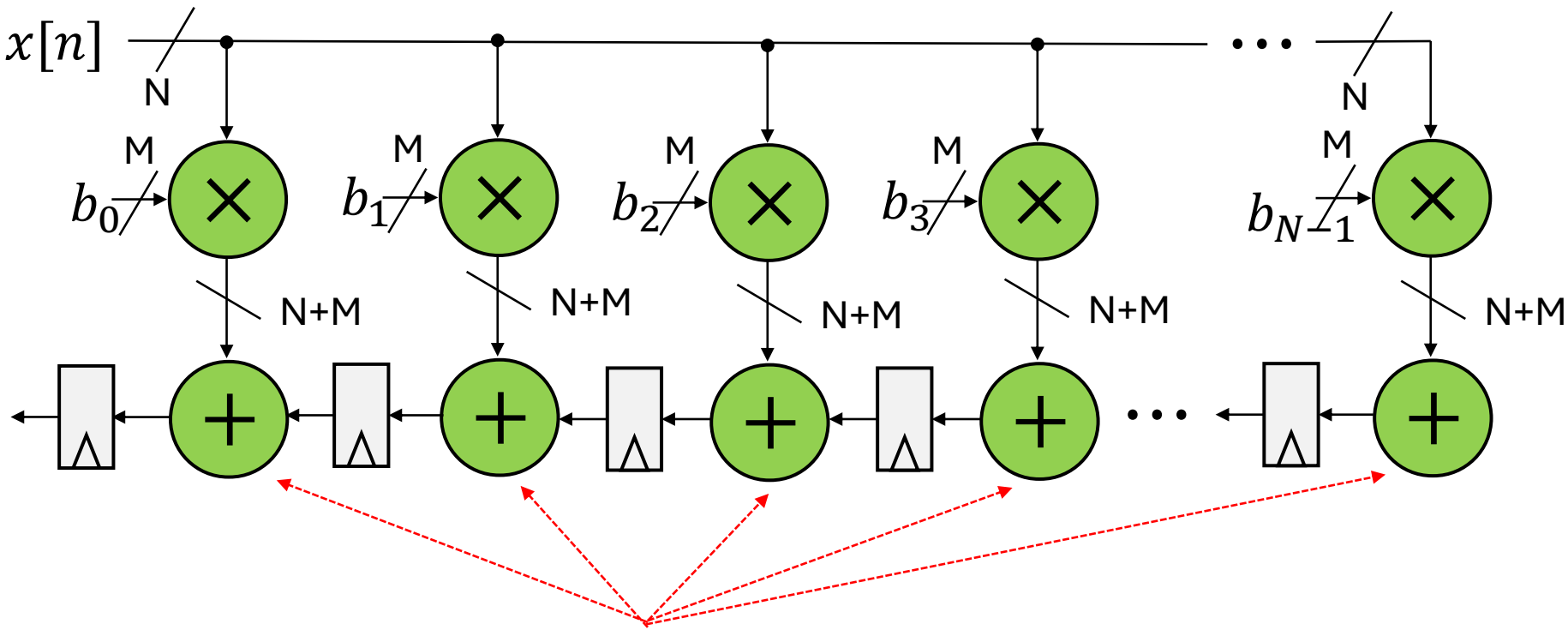
$$y[n] = \sum_{k=0}^{N-1} b_k \cdot x[n-k]$$



Much nicer critical path (worst propagation delay)

Bit Growth

$$y[n] = \sum_{k=0}^{N-1} b_k \cdot x[n-k]$$



Adding values that are N+M bits repeatedly grows the number of bits needed to not lose precision…will grow at between 1 bit per N and 1 bit per $\log_2(N)$! But this can grow large so there's ways to handle it

https://zipcpu.com/dsp/2017/07/21/bit-growth.html

Most FIR Filters (not all) are symmetric too.

- Depending on situation can double-up and feed back delayed signal

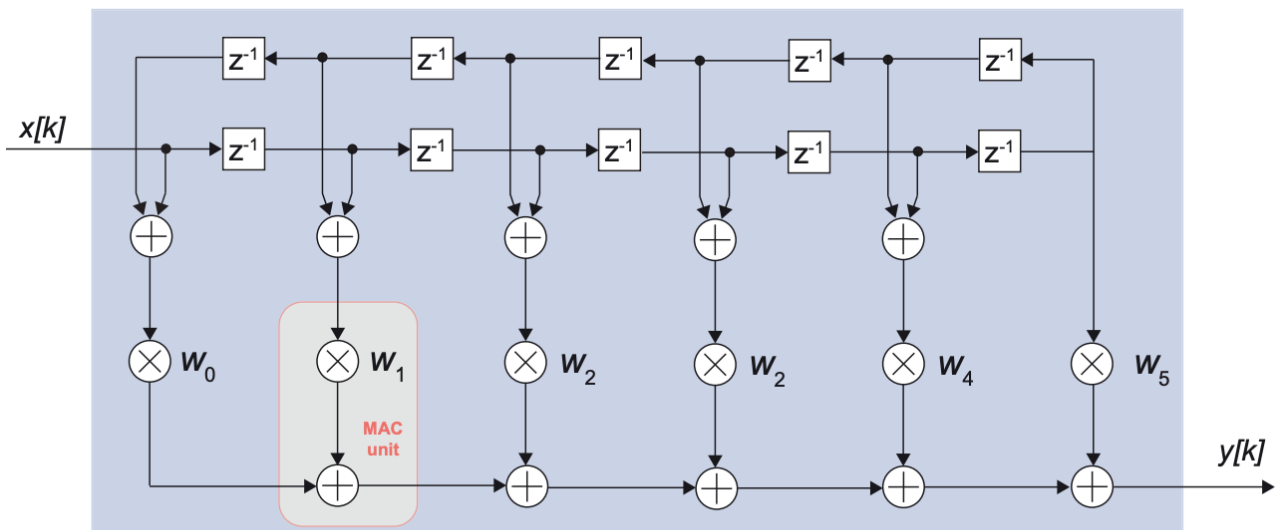


Figure 4.23: Example of a symmetric 11-weight FIR filter.

DSP Blocks?

- These IIR and especially FIR filters sure do have a lot of multiply-then-add operations going on…
- Remember those DSP blocks? That's why they're designed the way they are

DSP Blocks

- Mult-then-add is a common operation chain in many things, particularly Digital Signal Processing
- FPGA has dedicated hardware modules called DSP48 blocks on it
  - 150 of them on Urbana FPGA board
  - Capable of single-cycle multiplies
- Can get inferred from using * in your Verilog that isn't a power of 2:
  - x*y, for example, will likely will result in DSP getting used
  - May take a full clock cycle so would need to budget tiing accordingly

DSP48 Slice (High Level)

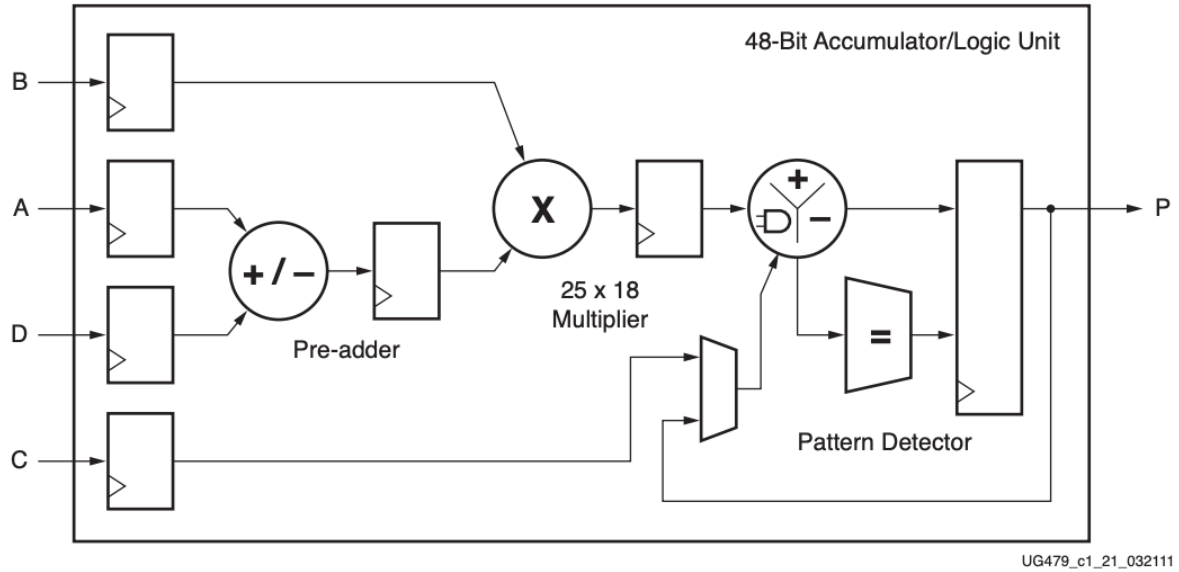


Figure 1-1: **Basic DSP48E1 Slice Functionality**

https://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf

DSP48E2 (Ultrascale +)

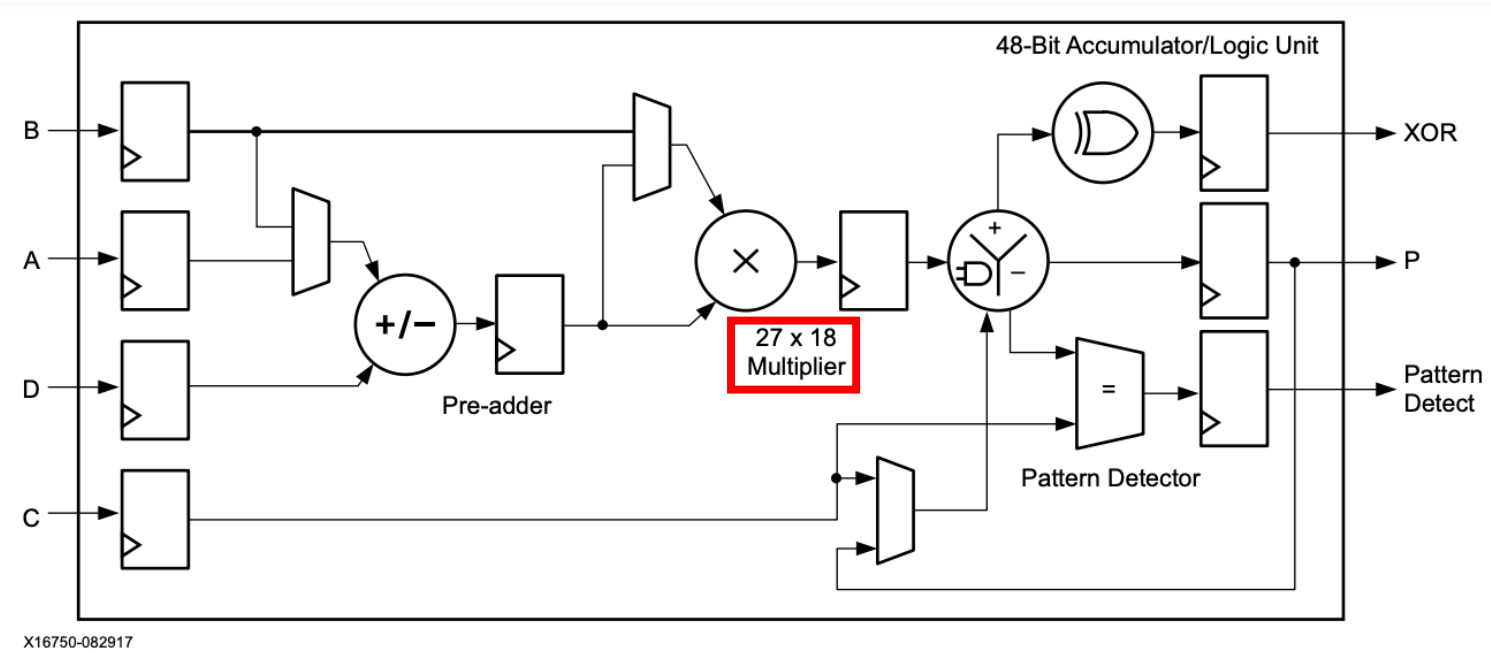
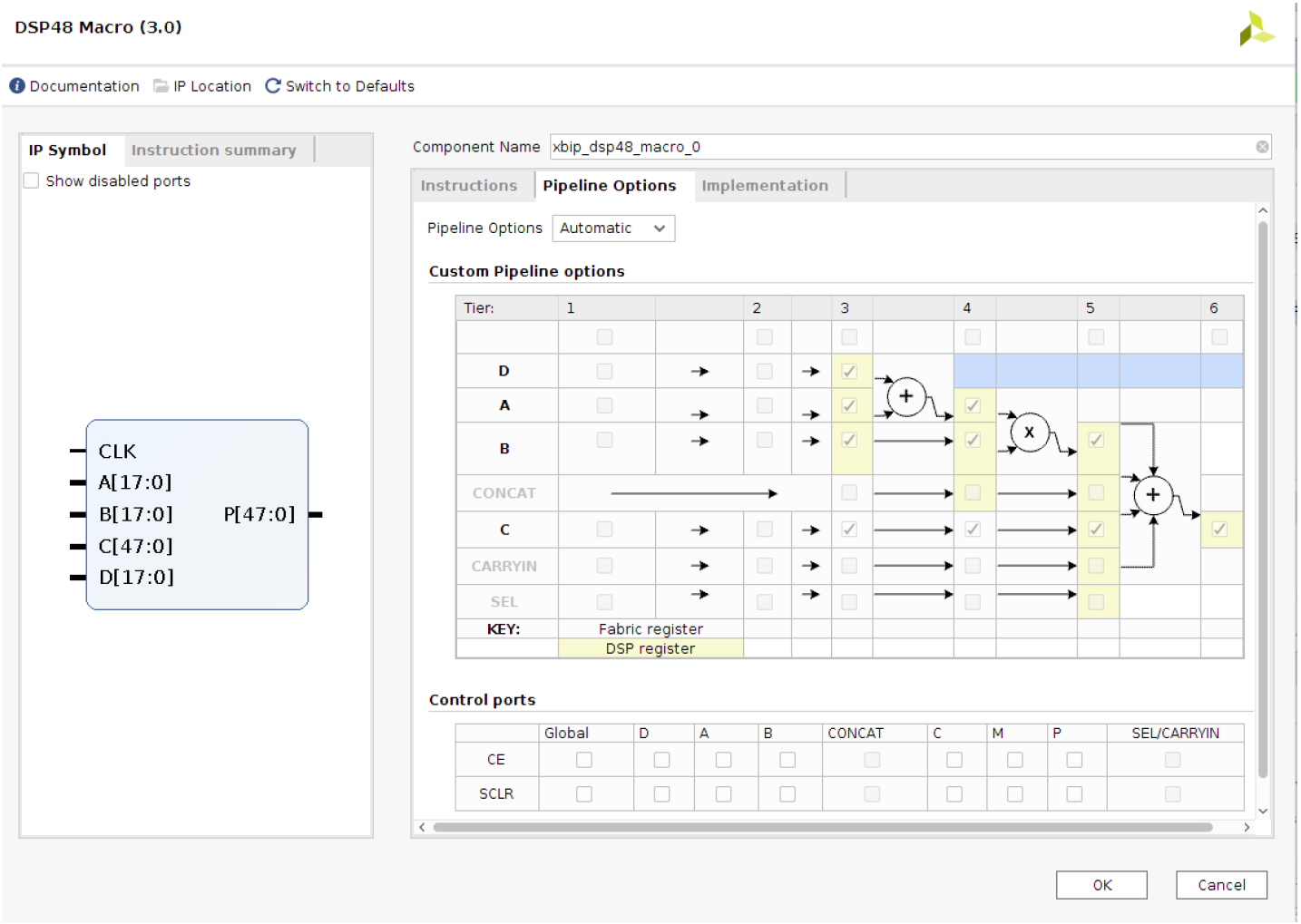


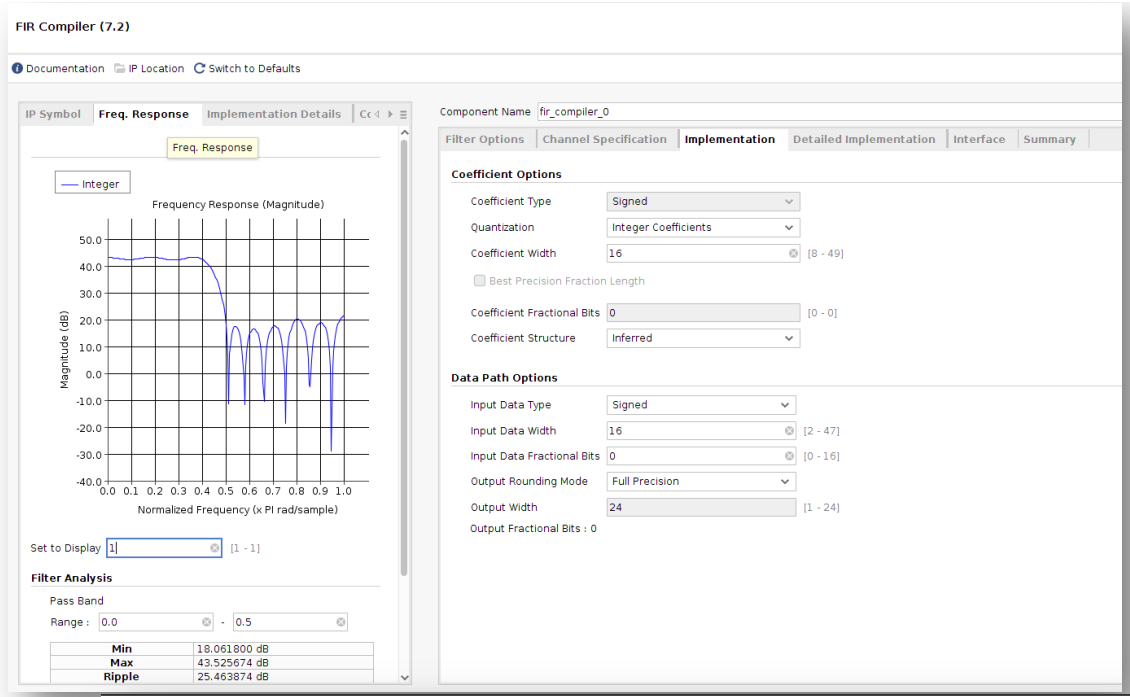
Figure 1-1: **Basic DSP48E2 Functionality**

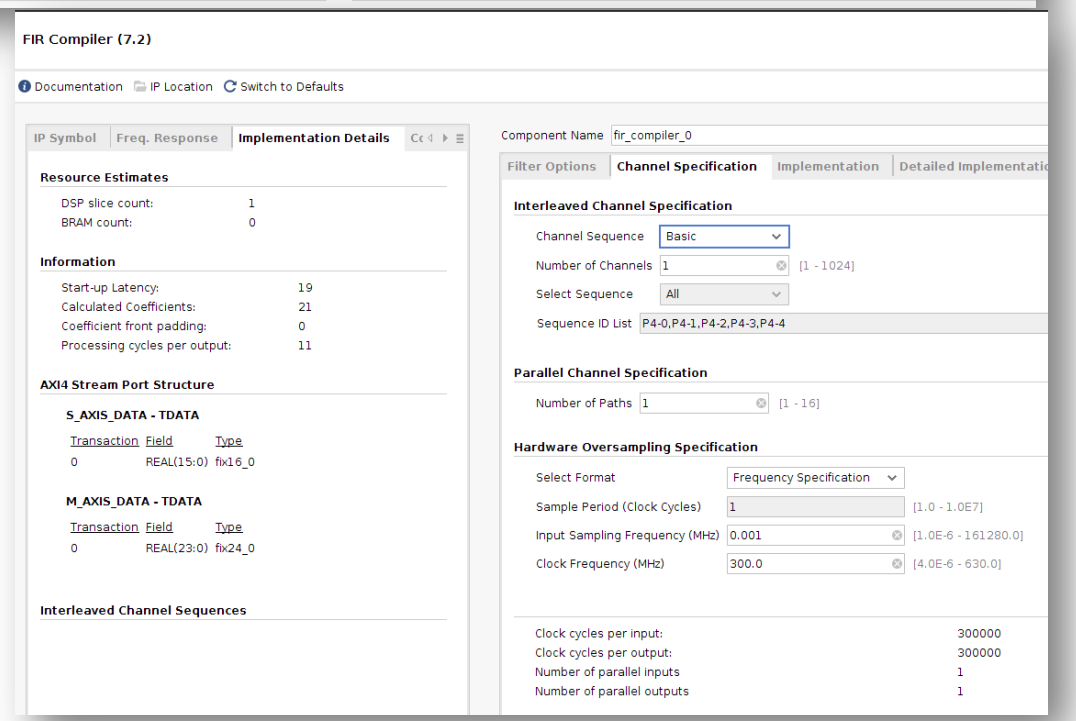
DSP Blocks


DSP48 Macro (3.0)
Documentation
IP Location
Switch to Defaults
IP Symbol
Instruction summary
Show disabled ports
CLK
A[17:0]
B[17:0]
C[47:0]
D[17:0]
P[47:0]
Component Name
xbip_dsp48_macro_0
Instructions
Pipeline Options
Implementation
Pipeline Options
Automatic
Custom Pipeline options
Tier:
1
2
3
4
5
6
D
A
B
CONCAT
C
CARRYIN
SEL
KEY:
Fabric register
DSP register
Control ports
Global
D
A
B
CONCAT
C
M
P
SEL/CARRYIN
CE
SCLR
OK
Cancel

FIR Wizard

- FIRs are so common, Vivado actually has some IP infrastructure to aid in designing them
- Can tune how pipelined vs. Iterative/FSM you want your FIR!
- Or use Python/numpy to determine coefficients





Project Idea (Smart Meter Listening)

- https://www.dailydot.com/debug/hacker-smart-meter-texas-snowstorm/
- https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7412105/

RAMBO

- https://arxiv.org/abs/2409.02292
- https://thehackernews.com/2024/09/new-rambo-attack-uses-ram-radio-signals.html