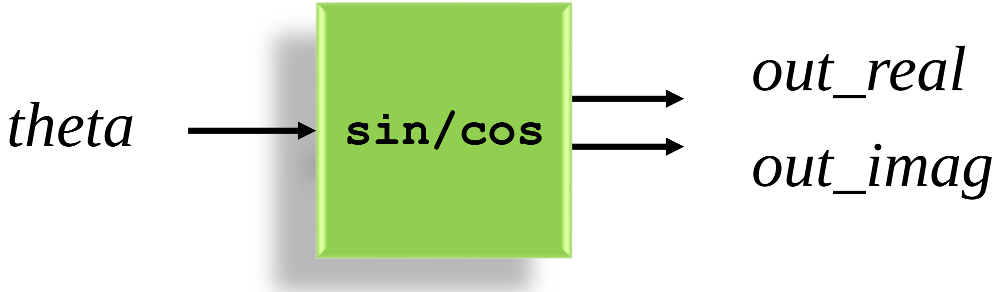
GENERATING COMPLEX FUNCTIONS

Generating Complex Functions

- Complex or "arbitrary" functions are not uncommon
- Examples
  - sin, cos, tan
  - tangent^{-1}
  - log
  - e^x
  - A/D converter correction values
  - RF mixer bias currents

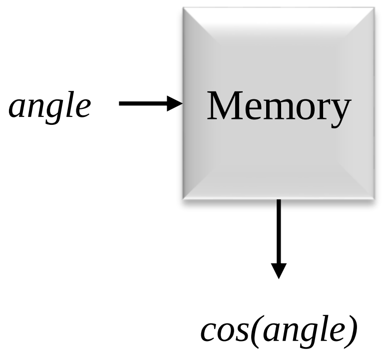


Generating Complex Functions
1) High-precision Numerical Calculations

- Almost certainly requires many clock cycles per calculation
  - 1–2 bits per clock cycle is common. In some cases, more bits/cycle are possible by adding hardware
  - Can regain *throughput* by parallel implementations
  - However *latency* is unavoidable
- Ex: CORDIC (Coordinate Rotation Digital Computer)
- Ex: polynomial expansions, etc.

Generating Complex Functions 2) Lookup Table

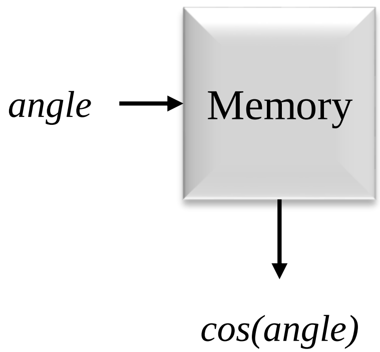
A. ROM array memory



- "Real" memory with address decoder, wordlines, bitlines, sense amplifiers, etc.
- Frequently available as macros from the standard cell vendor
- Could be mask-defined at manufacture, one-time programmable with fuses or anti-fuses, or flash non-volatile memory
- Generally compares better with very large tables since ROM cells are among the densest of all CMOS structures and there is a significant amount of overhead circuitry for a small memory

Generating Complex Functions 2) Lookup Table

B. Synthesized from standard cell combinational logic
   - The "memory" block is implemented by a highly-optimized netlist of combinational logic gates
   - Generally compares better with data that is less random (in an entropy information-theory sense) because it results in simpler and smaller logic equations
   - See notes describing ROM memories

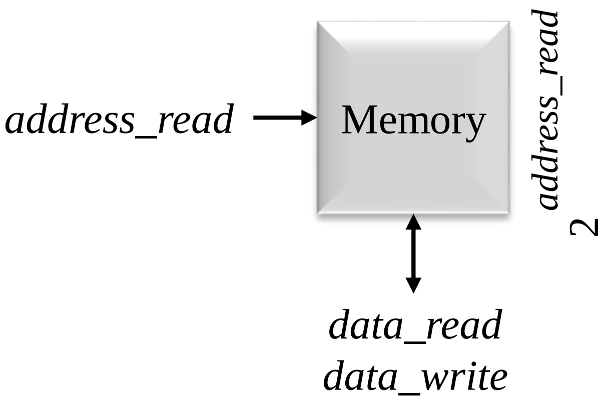


Input and Output Word Widths and Total Memory Size

- Total memory size

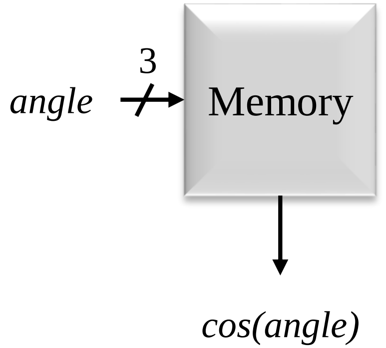
  $= 2^{address\_read\_width} \times data\_width$

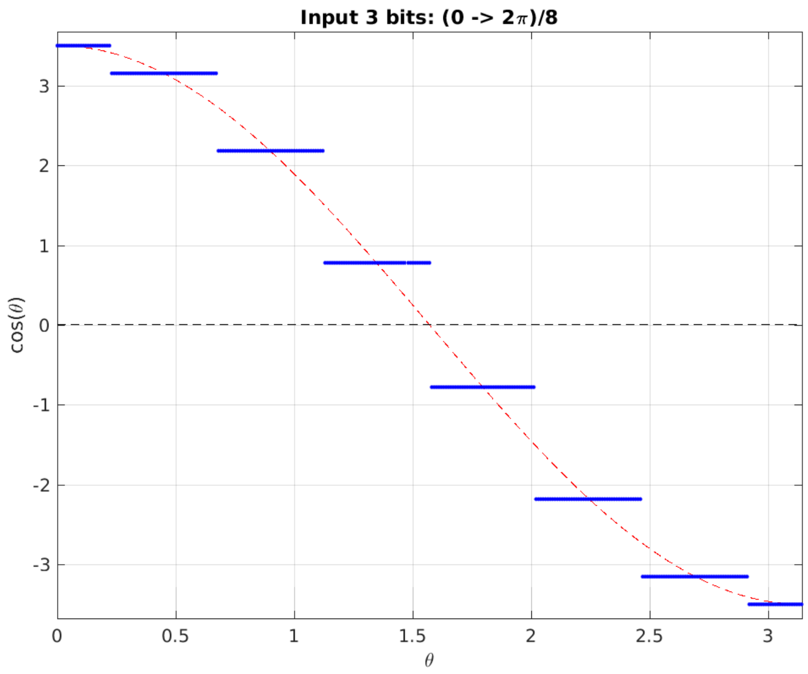
- The overall best word widths are a complex function of factors such as:
  - Overall system accuracy (e.g., SNR) requirements
  - Effect of word widths of particular signals on the overall system accuracy
  - Choice of numerical algorithms (e.g., table lookup and/or numerical methods)
  - Available SRAM and ROM technologies



Input and Output Word Width Effects

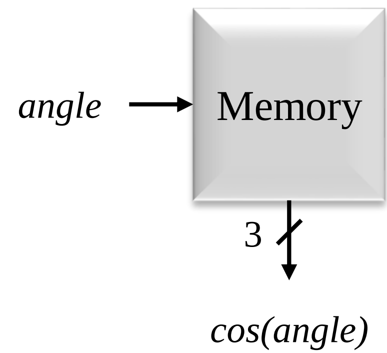
- Input word width
  - A narrow-word-width lookup table input increases the quantization granularity
  - Example: cos(theta[2:0])

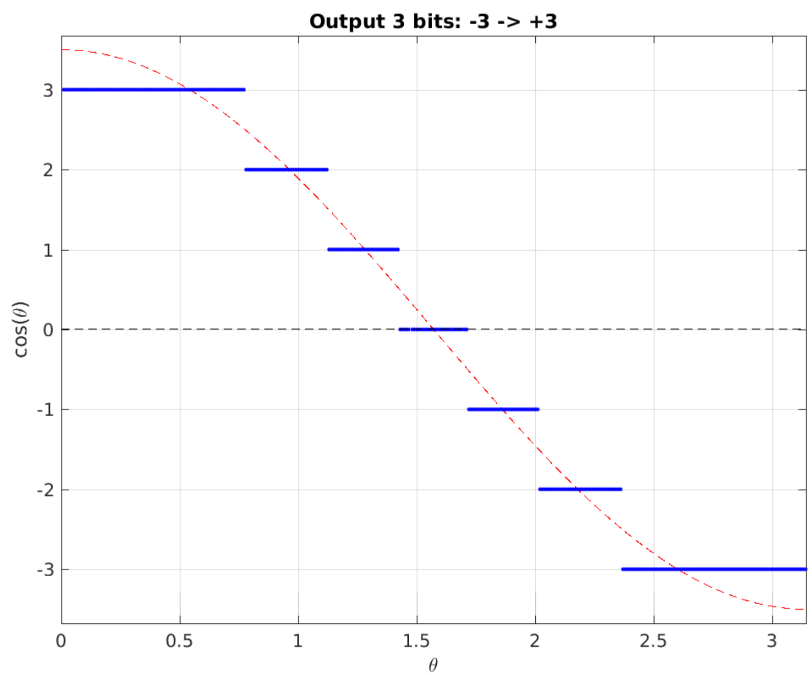




Input and Output Word Width Effects

- Output word width
  - A narrow-word-width lookup table input increases the quantization granularity
  - Example: y[2:0] = cos(theta)





matlab for previous plots

- copy, paste, and try it out

```
% wordwidth.m
%
% 2020/03/06  Written (BB)
%
% Bug: matlab isn't adding the title and axes labels unless those commands are
% copied & pasted by hand; I can't figure out why!


clear;

%--- Set these
PrintOn = 1;
x       = 0:0.01:pi;

%--- Main
figure(1); clf;
title('Output 3 bits: -3 -> +3');
xlabel('\theta');
ylabel('cos(\theta)');
Scale = 3.5;
y = Scale * cos(x);
plot(x, y, 'r--'); hold on;
y = round(Scale * cos(x));
plot(x, y, 'b.');
plot(x, zeros(1,length(x)), 'k--');     % black line
axis([0 pi -1.05*Scale 1.05*Scale]);
grid on;
if PrintOn  print -dpng quant.out.png;  end

figure(2); clf;
title('Input 3 bits: (0 -> 2\pi)/8');
xlabel('\theta');
ylabel('cos(\theta)');
Scale = 3.5;
y = Scale * cos(x);
plot(x, y, 'r--'); hold on;
x1 = x/pi;                % now [0 - 1]
x2 = x1 * 7;              % not ideal, [0 - 7]
x3 = round(x2);           %
x4 = x3/7*pi;             % [0 - pi]
y = Scale * cos(x4);
plot(x, y, 'b.');
plot(x, zeros(1,length(x)), 'k--');     % black line
axis([0 pi -1.05*Scale 1.05*Scale]);
grid on;
if PrintOn  print -dpng quant.in.png;  end
```

Lookup Tables with Cascaded Functions

- In many cases, computation is expressed or can be transformed into cascaded functions
- Example: The angle of a rectangular 2D vector = $\tan^{-1}(y/x)$
- A straightforward implementation using lookup tables would use a table for division followed by a table for $\tan^{-1}()$
- A better implementation would merge the cascaded functions into a single $\tan^{-1}(y/x)$ function implemented with a single memory
  - Assuming the intermediate result y/x is not needed elsewhere
  - In both cases, the input address is the concatenated *address* = $\{x, y\}$ or $\{y, x\}$; in fact, the bits from x and y can be mixed arbitrarily although the two examples here are certainly the clearest