A Real Number Model of a Phased Array Antenna

Rapid Adoption Kit

Daniel Cross, Sr Principal Solutions Engineer
Tim Pylant, Sr Architect

July 1, 2025

cadence®

Diagram For Intuitive Understanding

Transmit case

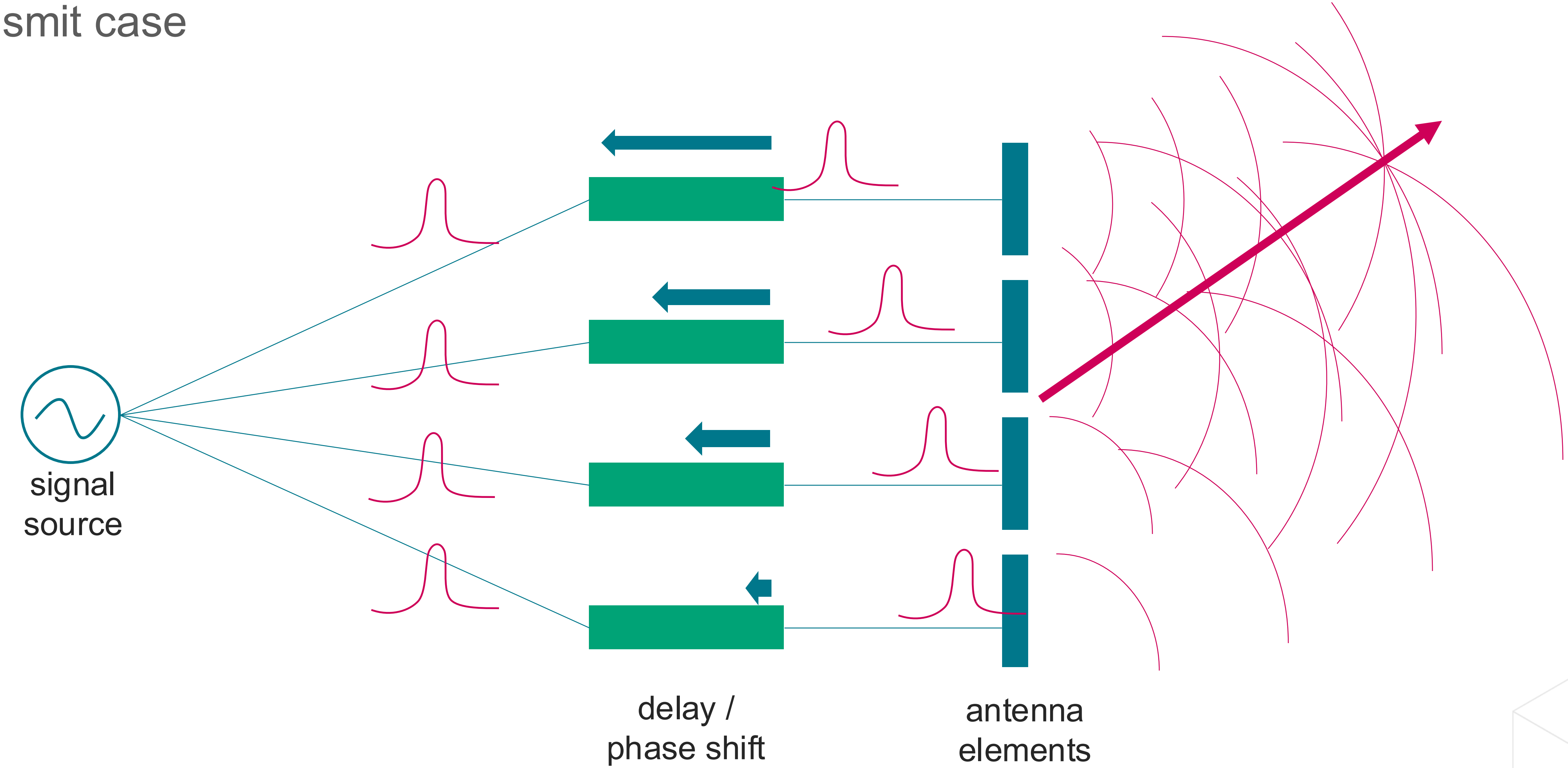


Diagram For Intuitive Understanding

Receive case – good

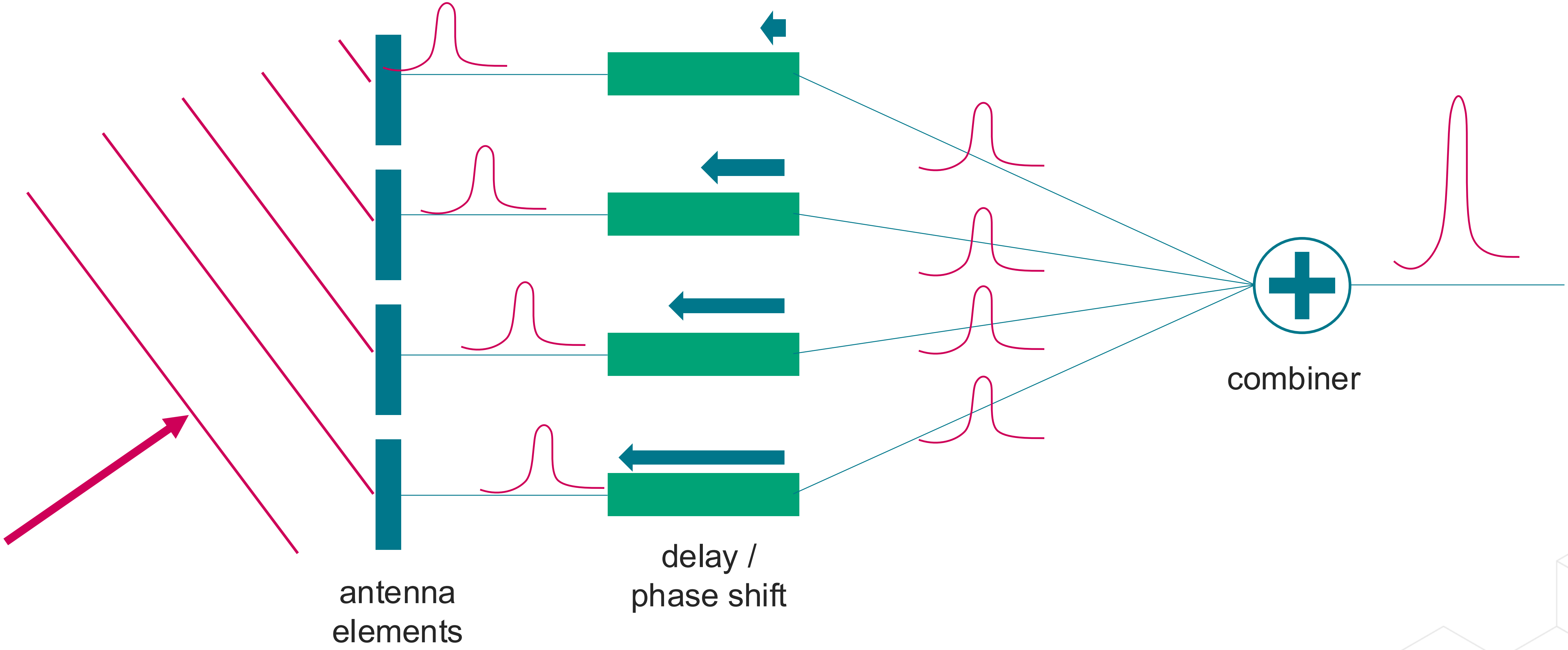
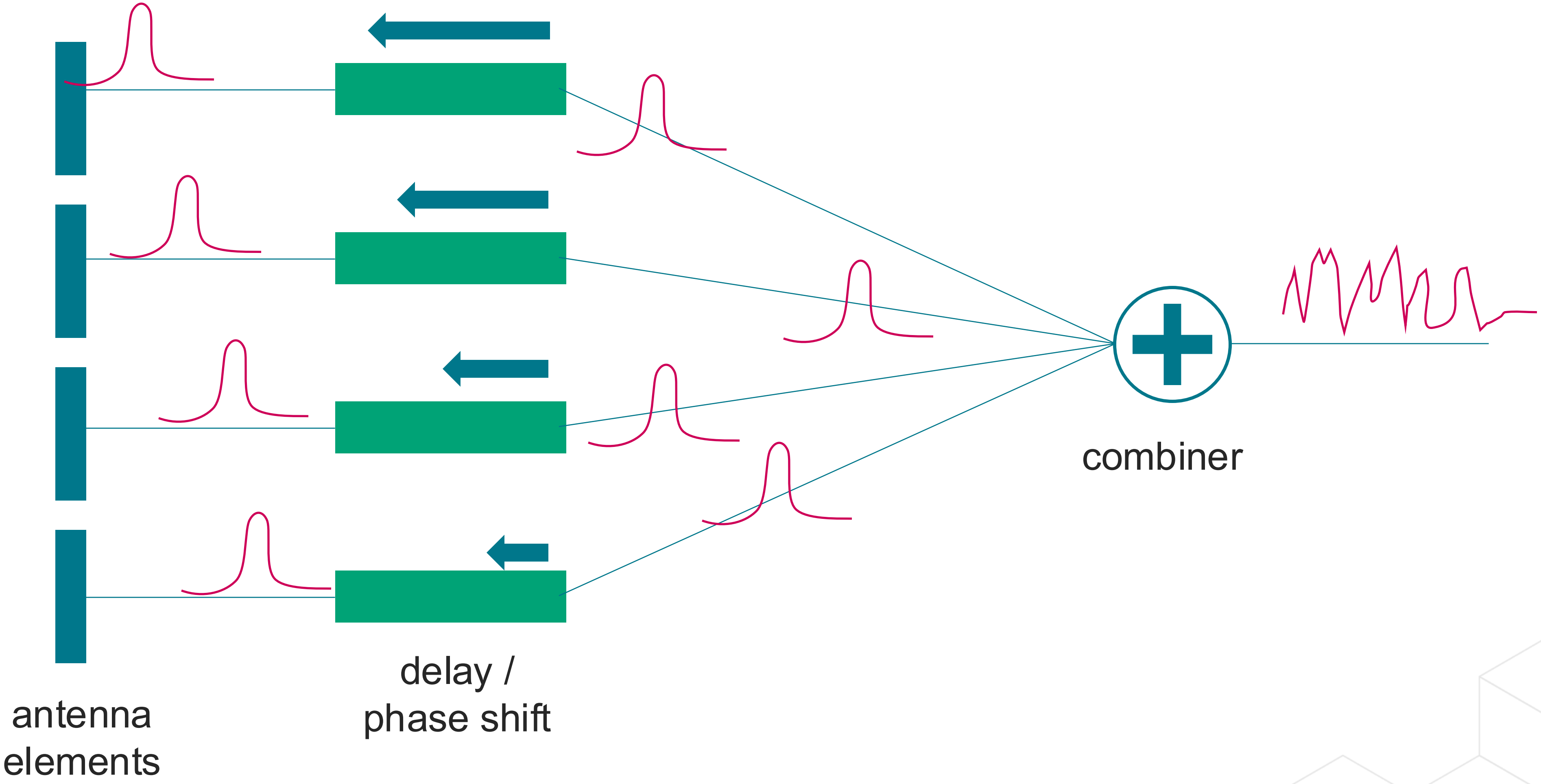


Diagram For Intuitive Understanding

Receive case – good



Mathematics

The behavior of the phased array is described as a diffraction phenomenon [1].

$$I(\theta) = [f(a, \sin\theta)]^2 \left[\frac{\sin\left(\frac{N}{2}\left(\frac{2\pi d}{\lambda}\sin\theta + \varphi\right)\right)}{\sin\left(\frac{\pi d}{\lambda}\sin\theta + \frac{\varphi}{2}\right)} \right]^2$$

a: size of antenna elements

N: Number of antenna elements

d: Spacing between elements

θ: Angle from normal to the array

φ: Relative phase shift between elements

λ: Wavelength

Since sin(π/2) = 1, the maximum occurs when the argument of the numerator is π/2. Also, we set d = *λ*/4 to make things easier.

cādence®

More Mathematics

$$\frac{\pi}{4} N \sin\theta + \frac{N}{2}\varphi = \frac{\pi}{2}$$

For transmit, we control φ (by setting the phase shift in the feeds) so we want the antenna model to solve for θ:

$$\theta = \sin^{-1}\left(\frac{2}{N} - \frac{2\varphi}{\pi}\right)$$

While for receive, we control the incoming angle θ (in the testbench) and we want the antenna model to solve for φ:

$$\varphi = \frac{\pi}{2}\left(\frac{2}{N} - \sin\theta\right)$$

Also see [3], p.540 for a slightly different treatment.

How do we model these mathematical equations on a net?

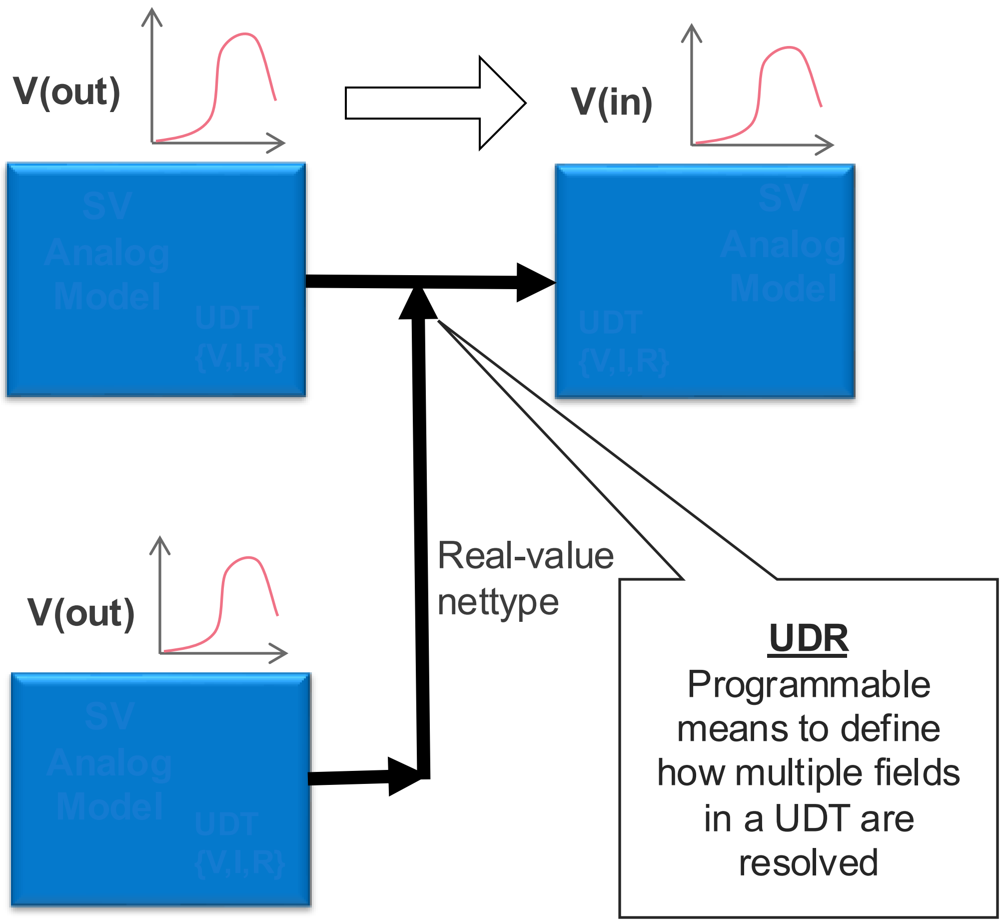
User Defined Nettypes Background

SystemVerilog IEEE 1800-2012 LRM

- User-Defined Types (UDTs)
  - Allows for single-value real nettypes
  - Keyword used: nettype
  - Allows for multi-value nets (multi-field record style)
  - It can hold one or more values (such as voltage, current, impedance) in a single complex data type that can be sent over a wire
- User-Defined Resolution (UDRs)
  - Functions to resolve user-defined types using keyword: with
  - Specifies how to combine user defined types
- Interconnect Nets
  - Types
    - Explicit: Type-less/Generic nets with keyword: interconnect
    - Implied: A Verilog(-AMS) net with keywords: wire, tri, wand, triand, wor, or trior
  - Used only for a net or port declarations

SystemVerilog User-Defined Nets

- User-Defined Nets can carry one or more values over a single net.
- Real values can be used to communicate voltage, current and other values between design blocks
- User-Defined Resolutions (UDR) functions are used to combine multiple outputs together.



cadence

Declaring User-Defined Nettype

- A SystemVerilog user-defined nettype without any resolution function can be declared as:

```
nettype T myNet;
```

Keyword — UDT — Nettype identifier

Example

```
module top;
 nettype  T  myNet;
 myNet w;
 assign w = T'{0.1, 0.2, 1'b1, 10};
 initial begin
     $display("Value of w -> %f => %p",$realtime, w);
  #1 $display("Value of w -> %f => %p",$realtime, w);
  #5 $display("Value of w -> %f => %p",$realtime, w);
 end
endmodule
```

```
// user-defined data type T
typedef struct {
  real     voltage;
  real     current;
  bit      field3;
  integer  field4;
} T;
```

UDT

Expected Output:

```
Value of w ->  0.000000 => '{voltage:0, current:0, field3:'h0, fileld4:x}
Value of w ->  1.000000 => '{voltage:0.1, current:0.2, field3:'h1, fileld4:10}
Value of w ->  6.000000 => '{voltage:0.1, current:0.2, field3:'h1, fileld4:10}
```

Assignment patterns

Declaring User-Defined Net with Resolution Function

- A user-defined SystemVerilog nettype with its resolution functions can be declared as:

```
nettype data_type  nettype_identifier with
                [package_scope|class_scope] tf_identifier] ;
```

  – nettype_identifier is the identifier you specify for the nettype.
  – [package_scope|class_scope] tf_identifier] can be a Cadence built-in resolution function or any typedef to the built-in real type

```
//Declaring a UDT nettype with UDR
nettype T  wTsum with Tsum;
```

```
// user-defined data type T
typedef struct {
  real field1;
  real field2;
} T;
```

UDT

```
// user-defined resolution function Tsum
function automatic T Tsum (input T driver[]);
  Tsum.field1 = 0.0;
  Tsum.field2 = 0.0;
  foreach (driver[i]) begin
    Tsum.field1 += driver[i].field1;
    Tsum.field2 += driver[i].field2;
  end
endfunction
```

UDR

cadence

User-Defined Nettype Example

Data Type and Resolution Function (As a Package)	Model

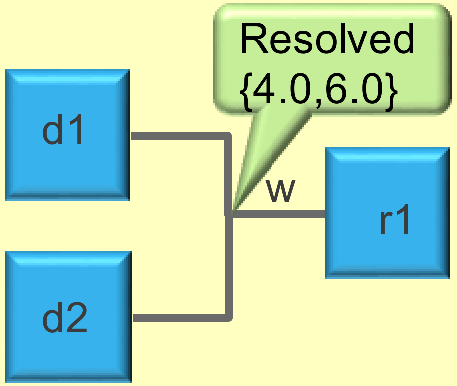
```
package temp_pkg;

// user-defined data type T
typedef struct {
     real field1;
     real field2;
}  T;

// user-defined resolution function Tsum
function automatic T Tsum (input T driver[]);
   Tsum.field1 = 0.0;
   Tsum.field2 = 0.0;
   foreach (driver[i]) begin
      if (driver[i].field1 !== `wrealZState)
         Tsum.field1 += driver[i].field1;
      if (driver[i].field2 !== `wrealZState)
         Tsum.field2 += driver[i].field2;
   end
endfunction

// nettype declaration datatype and resolution function
nettype  T  wTsum  with Tsum;

endpackage
```

UDT

UDR

Nettype

```
import temp_pkg::*;
module top;
   wTsum w;
   T  myvar;
   assign myvar = w;

   driver1 d1(w);
   driver2 d2(w);
   receiver1 r1(w);
endmodule

module receiver1 (input wTsum rec_1);
   always @(rec_1.field1, rec_1.field2)
  $display($time , ," sum = %f  flag = %f \n",
rec_1.field1, rec_1.field2);

endmodule

module driver1 (output wTsum dr_1);
   assign dr_1 = T'{1.0, 2.0};
endmodule

module driver2 (output wTsum dr_2);
   assign dr_2 = T'{3.0,4.0};
endmodule
```



cadence

Using UDNs for Modeling Beamformer

Special nettypes, definitions, and utilities needed for this example

RFpwrNet, radNet Definitions

This UDN (User Defined Nettype) provides a compact way to express signal values important for RF modeling.

```
typedef struct {
   real pwrdBm;
   real freq;
   real phase;
   real refZ;
}  RFpwrNetstruct;

nettype RFpwrNetstruct RFpwrNet;
```

This UDN has been developed just for this demonstration. It efficiently captures the information necessary to describe the radiated input or output from our antenna model.

```
typedef struct {
   real pwrdBWpm; // power emitted in dBW/m
   real freq;     // emitted frequency
   real theta;    // polar angle of main lobe
   real phi;      // azimuth angle of main lobe
}  radNetstruct;

nettype radNetstruct radNet;
```

Definitions and Utility Functions

Math Definitions

```
// definition of some basic math
//   constants and operations
`ifndef MATH_DEFINES
`define MATH_DEFINES
`define M_PI 3.14159265358979323
`define M_TWO_PI 6.28318530717958
`define M_PI_OVER_TWO 1.5707963267948966
`define M_BOLTZMANN 1.380649e-23
`endif
```

Utility Package

```
package utility_pkg;

function automatic real deg2rad (real degrees);
   return degrees * (`M_PI/180.0);
endfunction

function automatic real rad2deg (real rads);
   return rads * (180.0/`M_PI);
endfunction

endpackage
```

Models for Transmit

Antenna Model for Transmit

```
`include "mathDefs.sv"

module antArray import RFpwrNet_pkg::*; import radNet_pkg::*; #(parameter N=8)
(
   input  RFpwrNet rfFeed [N-1:0],
   output radNet   rfEmit
);

assign theta = $asin( (2.0/real'(N)) - (avgPhase/`M_PI_OVER_TWO) );

function automatic real avgPwr (RFpwrNetstruct inputs [N-1:0]);
function automatic real phiDiff (RFpwrNetstruct inputs [N-1:0]);
function automatic real calcAvgFreq (RFpwrNetstruct inputs [N-1:0]);

assign rfEmit = '{emittedPwr,avgFreq,theta,0};
```

Array of UDNs

Using assignment pattern for assignment to UDN

Phase Shift Model

```
`include "mathDefs.sv"

module phaseShift import RFpwrNet_pkg::*; #(parameter Bits=8) (
  input logic [Bits-1:0] shift,
  input RFpwrNet rfInput,
  output RFpwrNet rfOutput
);
parameter real lossdB = 0.1;

import utility_pkg::*;
real delPerBit = `M_PI / (2**(Bits-3));

phaseDelay = rad2deg(real'(shift*delPerBit));

assign rfOutput = '{outPwr,rfInput.freq,(rfInput.phase+phaseDelay),rfInput.refZ};
```

Using assignment pattern and UDT field references

Antenna Assembly

```
module txFE import RFpwrNet_pkg::*; import radNet_pkg::*;
   #(parameter N=8, Bits=8) (
     input  RFpwrNet rfInput,
     output radNet   rfOutput,
     input  logic [Bits-1:0] shiftCtl [N-1:0]
);
genvar k;
for (k=0;k<N;k++) begin
   phaseShift  #(.Bits(Bits)) x_phShift
      (.rfInput(rfInput),.rfOutput(rfFeeds[k]),.shift(shiftCtl[k])
   );
end

antArray #(.N(N)) x_antArray (.rfFeed(rfFeeds),.rfEmit(rfOutput)
);
```

Array of UDNs

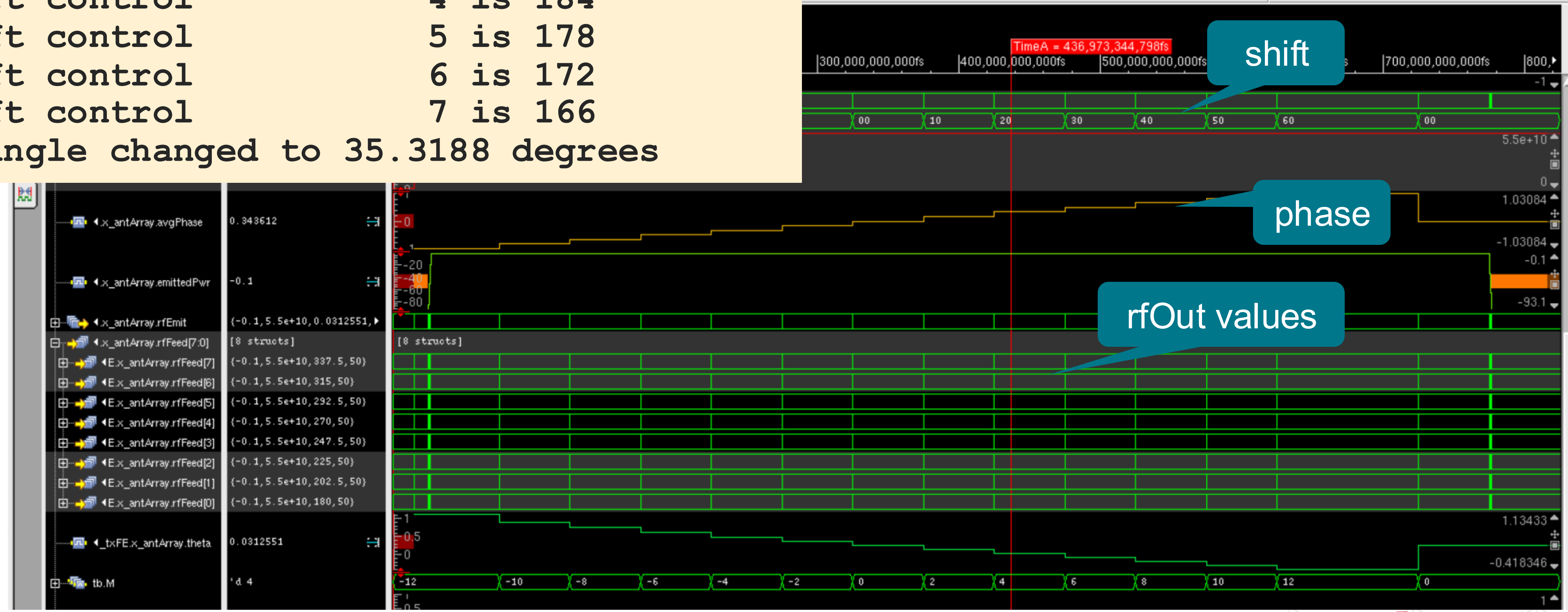
cadence

Testbench

```
module tb ( );
tbRFSource x_rfSrc ( );

txFE #(.N(N),.Bits(Bits)) x_txFE ();

function automatic shiftSets setShiftCtls (integer M);
   // sets digital delays to a gradient
endfunction

initial begin
  for (int k=0;k<12;k++) begin
      M = M + increment;
      shiftCtl = setShiftCtls(M);
      #(50us);
   end
end
```

Waveforms and Results

```
phase shift control          0 is 208
phase shift control          1 is 202
phase shift control          2 is 196
phase shift control          3 is 190
phase shift control          4 is 184
phase shift control          5 is 178
phase shift control          6 is 172
phase shift control          7 is 166
Transmit angle changed to 35.3188 degrees
```



Models for Receive

Antenna Model for Receive

```
`include "mathDefs.sv"

module antArray import RFpwrNet_pkg::*; import radNet_pkg::*; #(parameter N=8)
(
   output RFpwrNet rfFeed [N-1:0],
   input  radNet   rfCollect
);
assign theta = rfCollect.theta;
assign phiDiff = rad2deg(`M_PI_OVER_TWO * ((2.0/real'(N)) - $sin(theta)));
for (i=0;i<N;i++) begin
   assign rfFeed[i] = '{emittedPwr, rfCollect.freq, i*phiDiff, srcZ};
end

assign emittedPwr = rfCollect.pwrdBWpm;
```

Array of UDNs

Using assignment pattern and UDT field references

RF Combiner

```
module combiner import RFpwrNet_pkg::*; #(parameter N=8) (
  input  RFpwrNet rfInputs [N-1:0],
  output RFpwrNet rfOutput
);
typedef RFpwrNetstruct rfArray [N-1:0];
assign rfOutput = combineRFnets(rfInputs);
function automatic RFpwrNetstruct combineRFnets (rfArray inputs);
  for (int i=1;i<N;i++) begin
    pwrOut = W2dBm( abs(dBm2W(pwrOut) + dBm2W(inputs[i].pwrdBm) *
                    ($cos(deg2rad(phaseOut - inputs[i].phase))) );
    // Combining power at phase and degrees
    // Phase diff in radians, power at ratio
    phaseOut = (phaseOut + inputs[i].phase)/2;
    freqOut = (freqOut + inputs[i].freq)/2;
    refZOut = (refZOut + inputs[i].refZ)/2;
  end
  return '{pwrOut, freqOut, phaseOut, refZOut};
endfunction
```

Array of UDNs

Testbench

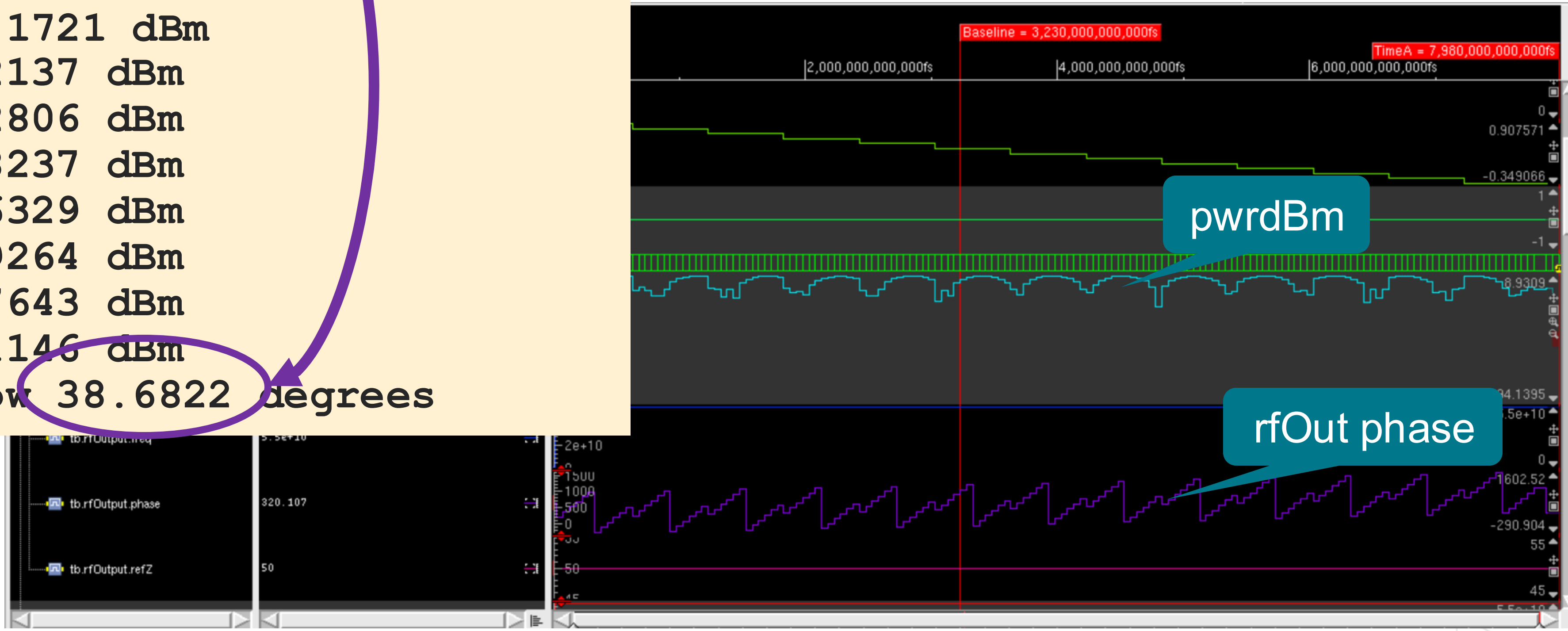
```
module tbRadSource import radNet_pkg::*; (
  input  real carrierFreq, power, theta,
  output radNet rfOut,
  input logic enable
);
 parameter phi = 0;  // azimuth not variable in this model
 always @ (posedge enable) begin
      rampStart = power - 31*rampStep; // set very low starting point
      thetaOut = theta;
 end
 always @ (negedge enable) begin
      rampStart = power; // set starting point at input power
      pwrInt = `wrealZState; // set to Z at end of ramp, it is OFF
      thetaOut = `wrealZState;
 end
 always @ (power) pwrInt = power;
 always @ (theta) begin
    if ((pwrInt !== `wrealZState) && (enable === 1'b1))
       thetaOut = theta;
 end
```

Testbench

```
module tb ( );
  tbRadSource x_radSrc (.carrierFreq ( rf_freq ), .power( rf_pwr ), .theta( targetTheta),
                        .rfOut ( rfIn ), .enable ( en ));

  rxFE #(.N(N),.Bits(Bits)) x_rxFE (
   .rfInput (rfIn), .rfOutput (rfOutput), .shiftCtl (shiftCtl));

initial
 for (int n=52;n>-24;n=n-6) begin
   targetTheta = deg2rad(real'(n));
   for (int k=0;k<12;k++) begin
      shiftCtl = setShiftCtls(M+(k*increment));
      #(50us) returnPwrs[k] = rfOutput.pwrdBm;
   end
   maxPower = returnPwrs.max();
   maxPwrIdx = returnPwrs.find_first_index() with (item == real'(maxPower[0]));
   maxPwrPhi = (M+(maxPwrIdx[0]*increment))*(`M_PI/32.0);
   maxPwrTheta = rad2deg(phi2Theta( -(M+(maxPwrIdx[0]*increment))*(`M_PI/32.0) ));
 end
```

Waveforms and Results

```
Target angle is now 40.0000 degrees
Output power is now -2.3203 dBm
Output power is now -0.8853 dBm
Output power is now -1.5603 dBm
Output power is now -8.0405 dBm
Output power is now -1.8007 dBm
Output power is now -9.1721 dBm
Output power is now 3.2137 dBm
Output power is now 6.2806 dBm
Output power is now 7.8237 dBm
Output power is now 8.6329 dBm
Output power is now 8.9264 dBm
Output power is now 8.7643 dBm
Output power is now 8.1146 dBm
Return peak angle is now 38.6822 degrees
```



Conclusion

Applications

A real-valued phased array antenna model such as the one presented here might be used for verification of systems such as these:

- Tracking radar.
- 5G communications.
- Ultrasound.
- Seismology.
- Radio astronomy.

Extensions

Additional features that could make the model even more expressive and useful are:

- Impedance effects.
- Antenna gain.
- Full radio front-end (amplifiers, mixers, ADCs, DACs).
- Wideband behavior (frequency variable element spacing).
- Signal modulation – frequency modulation is simplest.
- 2D array.

References

[1] "Diffraction from slits", https://en.wikipedia.org/wiki/Diffraction_from_slits, accessed 5/03/2022.

[2] "Phased array: Mathematical perspective and formulas", https://en.wikipedia.org/wiki/Phased_array#Mathematical_perspective_and_formulas, accessed 5/03/2022.

[3] Rao, N. N., *Elements of Engineering Electromagnetics*, 2nd Ed., Prentice-Hall, 1987.

[4] "Modeling RF Intermodulation Behavior with SystemVerilog Real Numbers", https://support.cadence.com/apex/ArticleAttachmentPortal?id=a1O3w00000AGwmNEAT&pageName=ArticleContent

cadence

Support and Feedback

- Support

  Cadence Learning and Support Portal provides access to support resources, including an extensive knowledge base, access to software updates for Cadence products, and the ability to interact with Cadence Customer Support. Visit https://support.cadence.com.

- Feedback

  Email comments, questions, and suggestions to content_feedback@cadence.com.