REFLECTARRAY DESIGN BASED ON RECTANGULAR PHOENIX CELL

Celeste Tan[1], Keenan Tan Han-Ming[2], Chia Tse-Tong[3], Tay Chai Yan[3]
[1]Raffles Institution (Junior College), One Raffles Institution Lane, Singapore 575954
[2]NUS High School of Math and Science, 20 Clementi Ave 1, Singapore 129957
[3]DSO National Laboratories, 12 Science Park Drive, Singapore 118225

Abstract

This paper presents a novel broadband single-substrate reflectarray design using rectangular Phoenix cells. The reflectarray is designed with a centre-fed horn to produce a broadside beam at 12.7 GHz for the vertical polarization. The rectangular Phoenix cell achieves a more linear phase change (compared to a square Phoenix cell), allowing for more accurate phase control. Full phase coverage can be achieved at the designed band 12.1-13.9 GHz. The measured results show that the proposed reflectarray achieved 30.1% efficiency.

1. INTRODUCTION

Printed reflectarrays are arrays of printed elements with varying sizes, each tailored to compensate for the different phase delays from the illuminating feed. They have the most salient features of both the traditional parabolic reflector antenna and microstrip array technology. [1] Due to their low profile, low mass and flat structures, reflectarrays can overcome mounting issues and conformal requirements. They are commonly used in micro-spacecrafts and satellites. [2]

One interesting element used for a reflectarray is the Phoenix cell, whose geometry exhibits a cyclic evolution, returning to its initial arrangement after a 360º phase cycle. [3] However, as far as the authors are aware of, the Phoenix cells used thus far are all squarish.

In this paper, we propose a rectangular Phoenix cell in Section 2 [4]. The most optimal prototype of the reflectarray using the given unit cell is determined in Section 3, and results are discussed in Section 4.

2. UNIT CELL CONFIGURATION

The proposed 9×12 mm^2 unit cell is shown in Fig. 1. It comprises the central rectangular patch and the outer ring, which are fixed. The size-variable ring grows in size from the inner rectangle patch to the outer ring, with a width:length ratio of 7:10. The unit cell is fabricated on a 0.5mm-thick FR4 substrate. The single layer unit cell of FR4 allows for simpler and cheaper fabrication. The substrate is separated from the ground plane by an air gap of 6mm. This spacing is optimized through simulation (see Appendix A, Fig. 1 for process) to achieve a slow-varying phase slope as shown in Fig. 2. The phase shift is determined by the growth variable *s*. The total phase shift is 360º which is typical of a Phoenix cell [3].

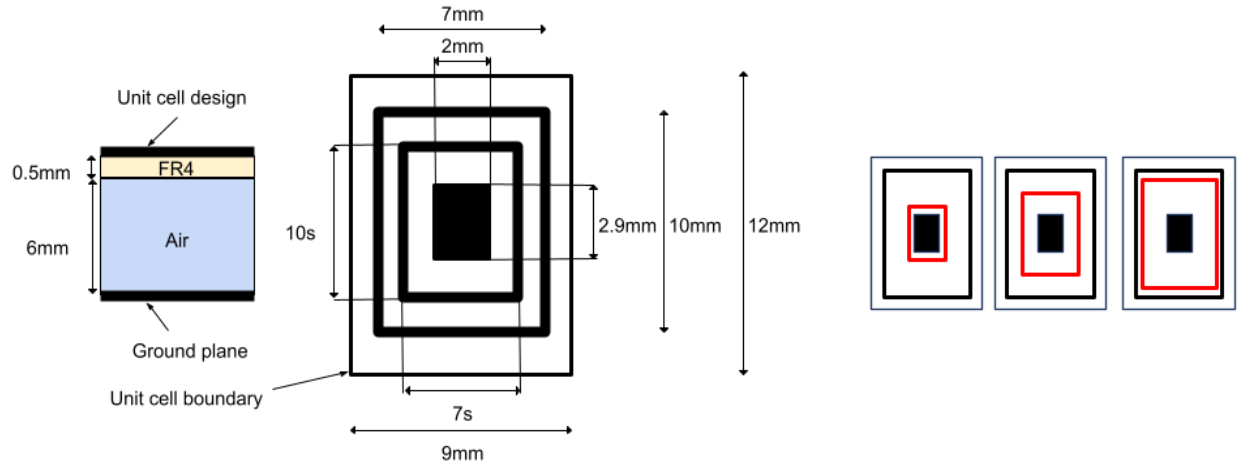


Fig. 1. (Left) Geometry of the proposed rectangular Phoenix cell. (Right) Illustration of growing ring (red).

To investigate the phasing behaviour of the unit cell, plane waves polarized vertically and horizontally at 12.7 GHz were incident normally on the unit cell using periodic boundary conditions in CST Studio Suite. The phase and magnitude of the reflected wave against the scaling parameter *s* (between 0 and 1) are plotted in Fig. 2, with “Rectangle long pol” and “Rectangle short pol” representing the polarized wave parallel to long side (vertical) and short sides (horizontal) of the rectangular cell, respectively.

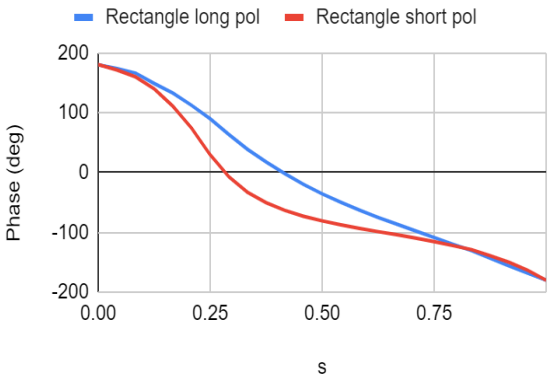


Fig 2. Phase responses of the rectangular Phoenix cell vs. scaling parameter, *s*, for 2 polarizations at 12.7GHz

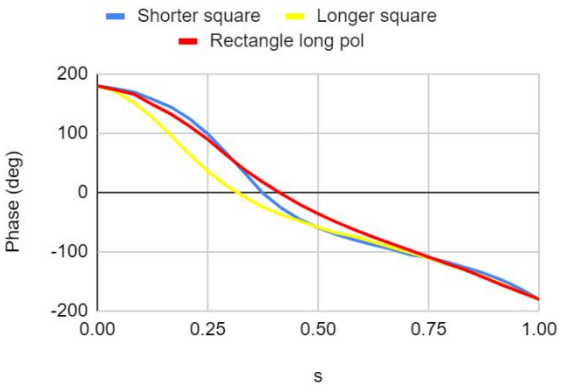


Fig 3. Phase responses of each Phoenix cell at 12.7GHz

The optimal polarization at 12.7GHz is thus along the longer side. We compared this unit cell design against the square Phoenix unit cell design [5] in Figure 3. In this work, the lengths of the squares are equal to the short (9mm) and long sides (12mm) of the rectangular design, respectively (see Fig. 4). All of them share close to the same phase range of 360° due to the Phoenix cell characteristic. The key characteristic of the rectangular Phoenix cell (for long pol) is its gentlest phase gradient, which is almost linear. The linearity allows for better and more accurate phase control as *s* changes. It will also mean greater tolerance if there is any slight fabrication error.

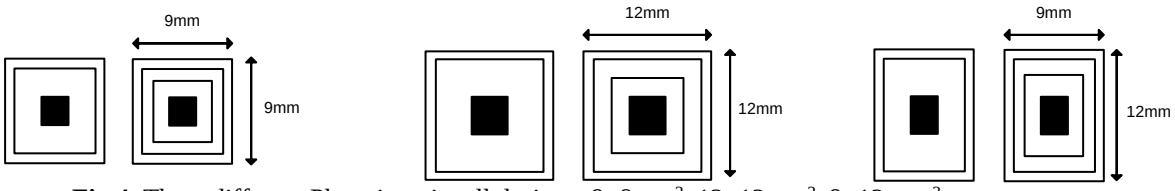


Fig 4. Three different Phoenix unit cell designs: 9×9mm^2, 12×12mm^2, 9×12 mm^2.

3. DESIGN OF REFLECTARRAY

Using the three unit cell designs in Fig. 4, 3 reflectarrays with the aforementioned constraints of the substrate's size were designed and simulated. Each approximately A4-size reflectarray is illuminated by a feed horn placed vertically above its geometrical centre at the optimal height of 179.2mm, determined from the -10dB point of the horn's main beam. An example of the reflectarray using the rectangular Phoenix cell is shown in Fig. 5.

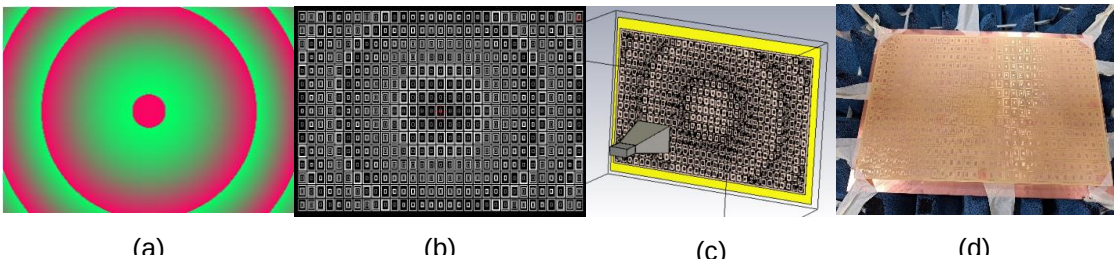


Fig 5. (a) Phase at each point calculated and represented by color (b) 2D form of best designed reflectarray accounting for the phase at each point (c) The reflectarray with all its components in CST (d) The milled and mounted prototype

The simulated gains and their corresponding efficiencies of the three reflectarrays are summarized in Table 1. The larger square unit cell array is much less efficient than the other two designs. This is because the phase distribution across the reflectarray with the larger square unit cell is a poorer approximation of the desired distribution. The smaller square unit cell and rectangular unit cell arrays share similar efficiencies.

In addition, the rectangular cell has an almost linear phase shift as well at 13.4 GHz (Appendix A, Fig 2). Therefore, the same rectangular Phoenix cell can potentially be used for either polarization, albeit at different frequencies.

Table 1. Simulated gains and computed efficiencies of reflectarrays using different unit cells.

	Smaller Square	Larger Square	Rectangle
Max Gain (dBi)	25.92	23.77	26.09
Efficiency (%)	33.34	20.33	34.71

Figure 6 shows that the reflectarray antenna using rectangular Phoenix cells has about 1.77 GHz bandwidth (or 13.6 %) for gain of 26.5±0.8 dB. At the designed frequency of 12.7GHz, the simulated realized gain of the reflectarray is 26.714 dBi, giving it a 40.1% efficiency.

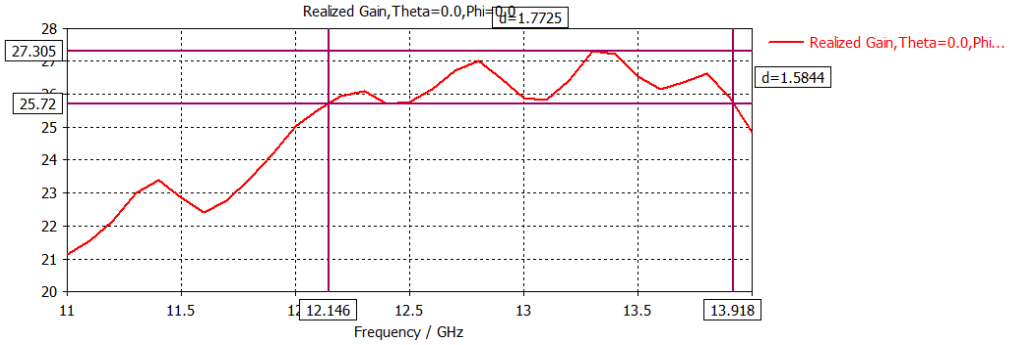


Fig 6. Gain vs. frequency for vertical polarization.

4. EXPERIMENTAL VALIDATION

The reflectarray using the proposed rectangular Phoenix unit cell shown in Figure 5 has been designed (see Appendix D for program) and fabricated. To create the air gap, 6 mm PLA spacers with a ±0.01 mm tolerance were used. Absorbers were placed around the reflectarray to prevent other reflections from the metal mount. The reflectarray was held down by masking tape. The support for the horn is made of wood, reducing the loss incurred from its blockage. The reflectarray is measured in the tapered chamber at TL@NUS (see Appendix B for photographs).

The measured results are compared against simulated results in Figure 7. The maximum gain in the measured azimuth and elevation (see also Table 2), differs by about 0.4dBi. While the difference is within measurement error, it could also be due mechanical alignment error between the reflectarray and the horn. We estimate that the reflectarray is misaligned in the broadside by about ±1°. Comparing the gain in the azimuth cut against the simulated gain, the gain difference is about 1.2dBi. This loss in gain can be seen in the higher measured first sidelobes.

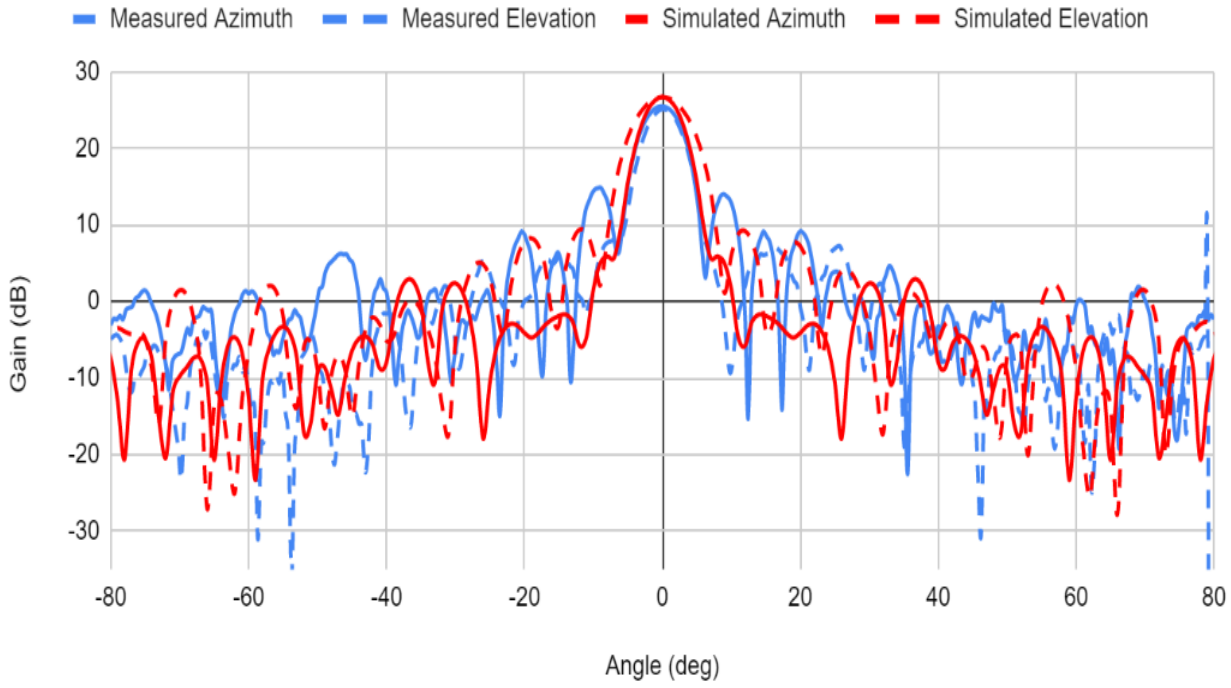


Fig 7. Measured radiation patterns for the reflectarray for azimuth and elevation cuts at 12.7GHz

Table 2. Realized gain of reflectarray prototype.

	Measured in Azimuth Cut	Measured in Elevation Cut	Simulated
Max gain (dBi)	25.478	25.096	26.714
Efficiency (%)	30.1		40.1

The measured gain-bandwidth is compared against the simulated results in Figure 8. The measured results are lower than the simulated results as expected. The gain loss is possibly due to the blockage by the horn and its support, and misalignment issues. The corresponding efficiency of the reflectarray against frequency is plotted in Figure 9 [1]. The maximum efficiency is at 13.6 GHz.

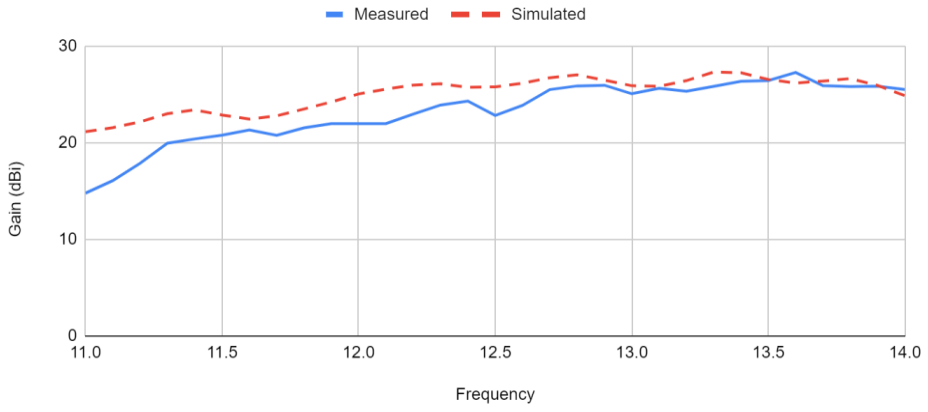


Fig 8. Gain against frequency of both the measured and simulated reflectarrays

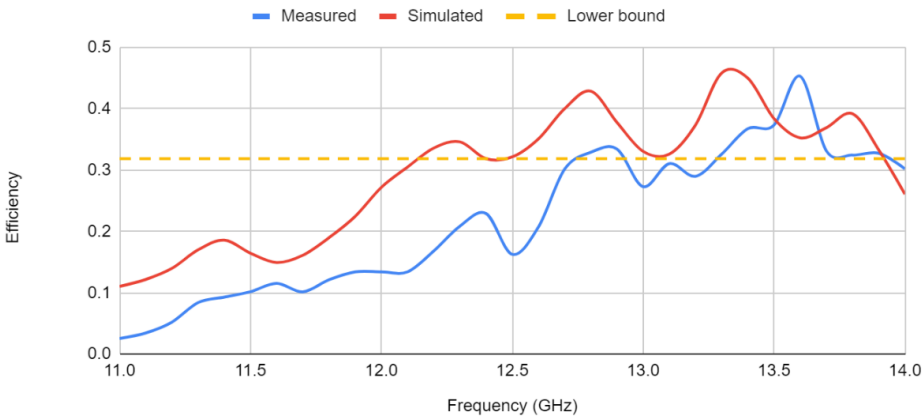


Fig 9. Efficiency against frequency for measured and simulated reflectarrays

5. LIMITATIONS

The reflectarray was designed for an existing horn that was optimal for 8.0-12.4 GHz, even though our frequency of interest was 12.7 GHz. The horn also had a circular shaped radiation beam at -10dB. Therefore, it will not be able to efficiently illuminate the rectangular reflectarray. To improve the illumination efficiency, the horn can be offset from the centre of the reflectarray so that its elliptically shaped beam at the plane of the reflectarray will illuminate a wider portion of the reflectarray.

Due to time constraints and availability of the measurement resources (MP and facility), we only had half a day to set up and measure the reflectarray. As the feed horn was not aligned accurately in the broadside of the reflectarray (feed horn was "centered" using a plumbline which was susceptible to human error), the horn and reflectarray was offset by about 1°, causing a shift in the main beam of the reflectarray. Furthermore, the reflectarray was not secured enough, such that it suffered a noticeable shift every time it was moved or mounted. The measurement was thus compromised.

6. FURTHER WORK

After fabricating the reflectarray, we did an analysis on the optimal height of the horn to maximize efficiency of the antenna [6]. We modeled the feed pattern F of our horn as a $cos^q\theta$ function. Approximating the feed radiation pattern to be perfectly circular, we got a q_e and q_h of 8.29 (in the E- and H- planes), where the horn has -10dB at θ=29.5°. We then plotted a graph of efficiency against R/λ by calculating its spillover efficiency and illumination efficiency (see Appendix E for program written).

Our fabricated height of 179.2mm (or R=7.59λ) was obtained by using the -10dB of the horn's radiation pattern at the long edge of the reflectarray to minimize spillover. The optimal height with maximum efficiency is 66.3% using the written program was found to be 247.8mm (or R=10.5λ) as shown in Figure 10. For our chosen height, the expected efficiency is 59.9%. Thus, the efficiency of the proposed reflectarray antenna could be improved upon if the horn was moved further away from the array.

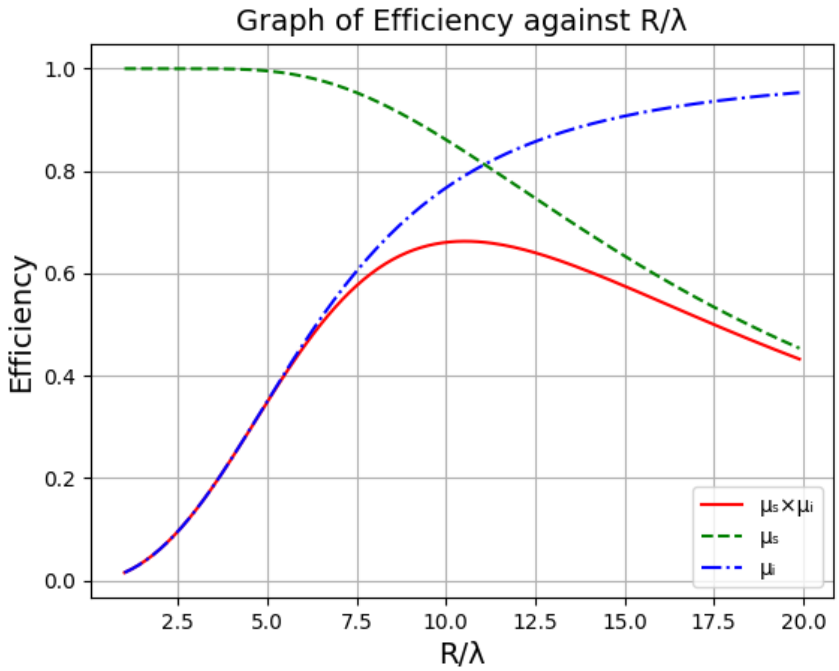


Fig 10. Efficiency against height (μ_i: illumination efficiency, μ_s: spillover efficiency).

7. CONCLUSION

A broadband reflectarray design using a rectangular Phoenix cell has been designed. The reflectarray has a gain of 26.5±0.8 dB from 12.1 to 13.9 GHz. The Phoenix cell achieves an almost linear phase shift, allowing for greater phase control and increased fabrication tolerance. The designed reflectarray achieved an efficiency of 30.1%, compared to the simulated 40.1%. The measured results are not ideal, partly caused by inefficient illumination and alignment issues which can be further improved on.

REFERENCES

[1] J. Huang and J. A. Encinar, Reflectarray Antennas, by Institute of Electrical and Electronics Engineers, John Wiley & Sons, 2008.
[2] Adel, S., & Hammad, H. Modified Phoenix cell for microstrip reflectarray antennas, 2012.
[3] Chao Tian, Yong-Chang Jiao, and Weilong Liang, "A Broadband Reflectarray Using Phoenix Unit Cell," Progress In Electromagnetics Research Letters, Vol. 50, 67–72, 2014.
[4] Lu Guo, Peng-Khiang Tan, and Tan-Huat Chio, "Single-Layered Broadband Dual-Band Reflectarray With Linear Orthogonal Polarizations," EEE Trans. Antennas and Propag., vol. 64, no. 9, pp. 4064-4068, Sept. 2016.
[5] Ruyuan Deng, Fan Yang, Shenheng Xu, and Maokun Li, "Design of a Single-Layer Dual-Band Reflectarray Using Phoenix Elements," 2015 Asia-Pacific Microwave Conference (APMC), 2015, pp. 1-3, doi: 10.1109/APMC.2015.7413393.
[6] Zebrowski, M., "Reflectarray Antennas Illumination and Spillover Efficiency Calculations for Rectangular Reflectarray Antennas," *High Freq. Des.*, Vol. 1, 28-38, Dec. 2012.

ACKNOWLEDGEMENTS

We would like to acknowledge and give our warmest thanks to our mentors, Dr Chia Tse Tong and Dr Tay Chai Yan, from DSO National Laboratories, for making this project possible. Their invaluable guidance and support during the process made us stay focused and motivated, clarifying any doubts we had about the topic.

We would also like to thank the DSO National Laboratories for providing us with the opportunity and experience, and Temasek Laboratories (TL) @ NUS, especially Mr Tan Peng Khiang, for helping us with the use of the lab resources for fabrication and measurement of our fabricated antenna.

APPENDIX A: Antenna design process

In this section, additional information or factors considered in designing the antenna is provided. Supplementing graphs of lesser importance are placed here, followed by a page number that the reader should reference for context on the graphs and how they factor into design.

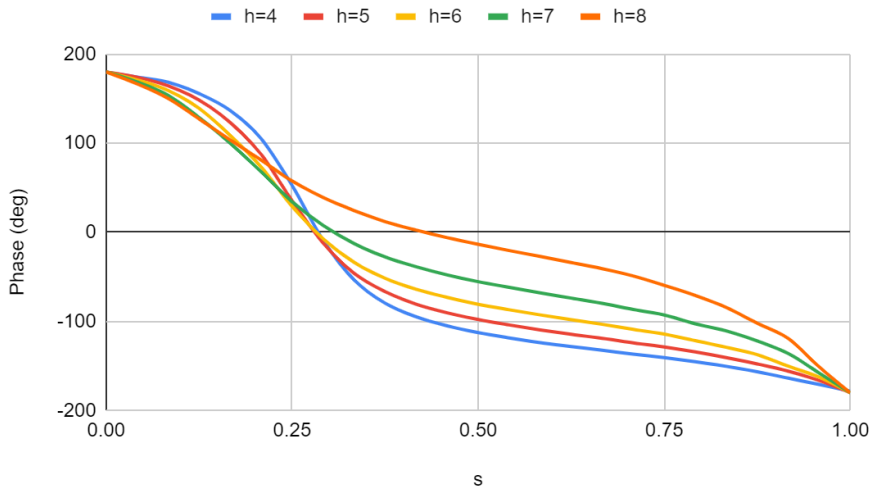


Fig 1 (Page 1). Varying the air height to obtain optimal phase curve shape

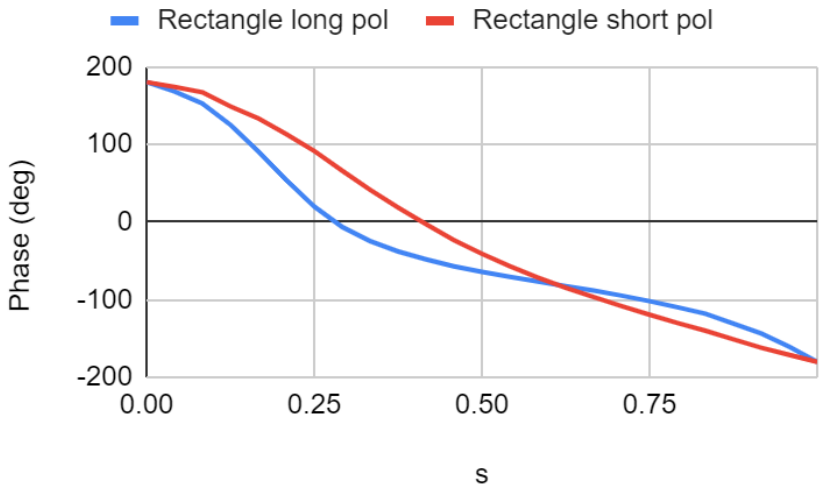


Fig 2 (Page 2, Page 3). Phase responses of the rectangular Phoenix cell for each sub-wavelength element at *13.4GHz* for both polarizations.

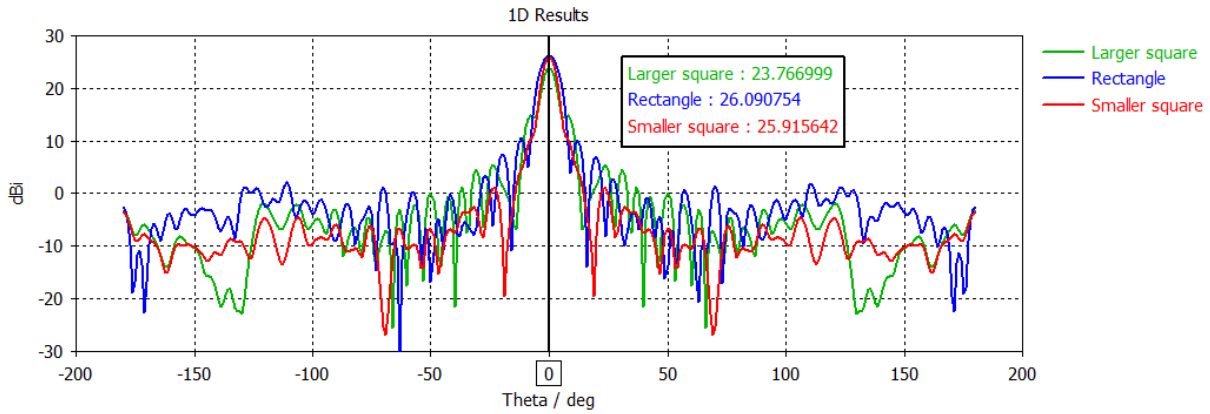
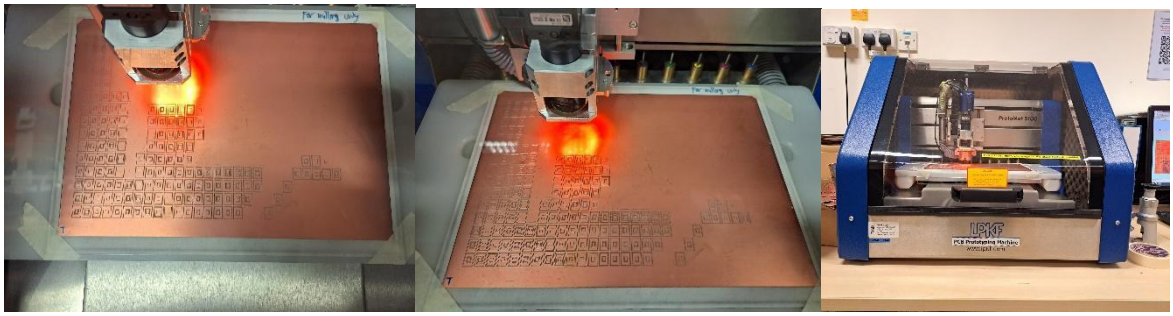


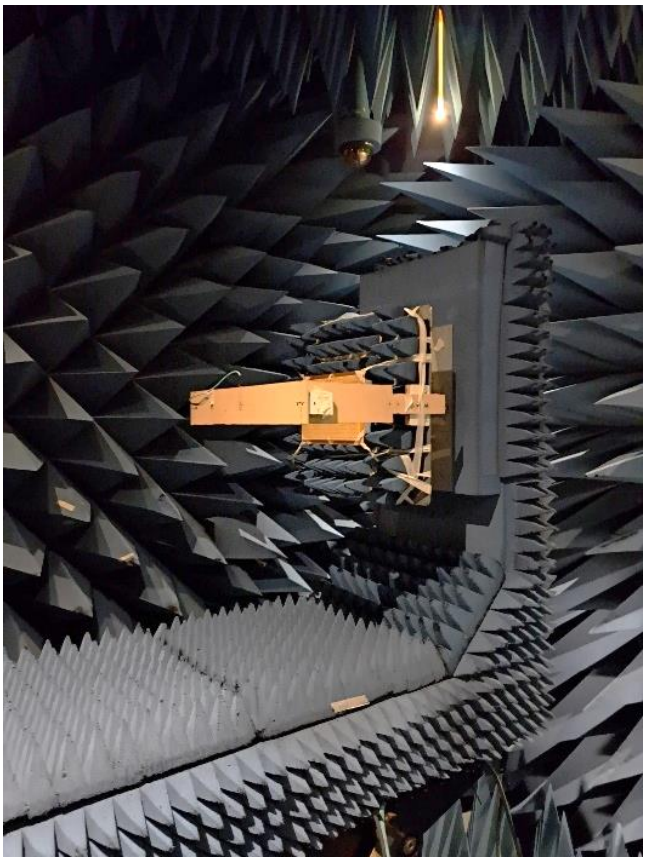
Fig 3 (Page 3). Gain of all the different unit cell array in azimuth cut (Table 1)

APPENDIX B: Information on reflectarray prototype

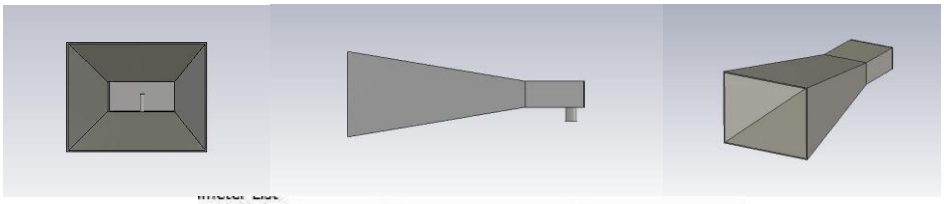
In this section, photos of the fabricated reflectarray prototype in construction or measurement are placed to give readers a better understanding of the design. Information on the feed horn is placed here as well.



1. Reflectarray milled out of FR4 in Temasek Laboratories @ NUS.



2. Reflectarray setup mounted in the antenna measurement chamber at Temasek Laboratories @ NUS



Name	Expression	Value
taper_angle	= 10	10
horn_length	= 30	30
wall_thickness	= 1	1
waveguide_width	= 10	10
waveguide_height	= 5	5
h	= 30	30
XMAX	= 50	50
zmax	= 100	100
<new parameter>		

3. Different perspectives of the feed horn used and its dimensions in CST

4. Assembly parts of the prototype (Left) 3d printed 6mm spacers (Middle & Right) Mounted reflectarray. Note that absorbers were placed on the metal support that is larger than the reflectarray.

APPENDIX C: Code developed for auto-generation of .dxf reflectarray files of proposed design

There are 9 parts to this section explaining the code written in Python used in the reflectarray design.

1. Automation to generate all five files together
2. Obtaining the size of the array, the height of the horn, and the functions to determine phase at each position of the reflectarray
3. Reading and formatting the data of S11 graph
4. Testing the different interpolation types
5. Setting up the writing of the scripts for LibreCAD to create .dxf files
6. Automation for generation of the .dxf files
7. Import needed libraries
8. Using the functions written above to read data of S11 curve, and interpolate it using direct interpolation (which gave the best result)
9. Using the graph generated and functions above to get the phase at each point of the reflectarray and adjust the drawing accordingly

Each section contains the code used as well as comments briefly explaining what each section does. When pieced together, the entire program should run using a simple iteration by changing the type_of_gen variable from 0 to 5, with 0 producing the overall.dxf with all the components in one .dxf file.

1. Automation to generate all five files together

```
actual = True #change to false if maximise array
phi_0_angle = 30.368 * math.pi / 180 #smaller angle
phi_90_angle = 27.8 * math.pi / 180 #larger angle

file_names = ["overall", "inner_ring", "inner_ring_subtract", "outer_ring", "outer_ring_subtract", "inner_rect"]


file_name = file_names[type_of_gen]
folder_name = "<insert folder name>"
printfile = open(f"{folder_name}/prints.txt", "w") #to record the phase and ss of each positions
info_file  = open(f"{folder_name}/info.txt", "w") #to record the height, frequency and everything that is printed
outfile = open(f"{folder_name}/{file_name}.txt", "w") #for generating the actual commands
```

2. Obtaining the size of the array, the height of the horn, and the functions to determine phase at each position of the reflectarray

```
thetaB = 0
phiB = 0
freq = 12.7*pow(10, 9)
wavelength = 3*pow(10, 8)*pow(10, 3)/freq
k0 = 2*math.pi/wavelength
UCsize_length_x = 9 #mm
UCsize_length_y = 12 #mm
a4size_x = 297 #mm
a4size_y = 210 #mm

def determine_across_down(fullsize_x, fullsize_y):
    down = fullsize_y // UCsize_length_y - 2
    horn_height = (down * UCsize_length_y / 2) / math.tan(phi_90_angle)
    max_10db_x = horn_height * 2 * math.tan(phi_0_angle)
    across = int(max_10db_x // UCsize_length_x) if actual else fullsize_x // UCsize_length_x - 2 #-2 is taken away
because of gap already beside
    return across, down, horn_height

across, down, horn_height = determine_across_down(a4size_x, a4size_y)
```

```
center_x = across*UCsize_length_x/2
center_y = down*UCsize_length_y/2
source_z = horn_height

def getDi(xi, yi):
    # assuming xi, yi is the distance from the center in cartesian x and y
    di = math.sqrt(abs(xi)**2+abs(yi)**2+source_z**2)
    return di

def get_phase(x, y):
    xi = UCsize_length_x/2+x*UCsize_length_x-center_x
    yi = UCsize_length_y/2+y*UCsize_length_y-center_y
    di = getDi(xi, yi)
    phase = k0*(di-(xi*math.cos(phiB)+yi*math.sin(phiB))*math.sin(thetaB))
    return phase, xi, yi
```

3. Reading and formatting the data of S11 graph

```
#reading data

def read_data():
    infile = open("<insert results file>.txt", "r")
    data = infile.read().replace("","").strip().split("\n")
    infile.close()
    x_values = []
    y_values = []
    # data[0] = data[0][3:]
    correction = False
    for index, item in enumerate(data):
        x, y = item.split("\t")
        x_values.append(float(x))
        if index == 0:
            first = float(y)
        if index != 0 and abs(float(y) - float(prev_y)) > 100: #accounting for phase jump
            correction = True
        if correction:
            y_values.append(float(y)-first)
        else:
            y_values.append(float(y)-first+360)
        prev_y = y
    # print("original ss:", x_values)
    # print("original phase:", y_values)
    return x_values, y_values
```

4. Testing the different interpolation types

```
def linear():
    plt.plot(x,y,'ro')
    plt.plot(x,y, 'b')
    plt.title("Data set and linear interpolation")
    plt.show()

def direct():
    tck = interpolate.splrep(x, y, s=10)
    xfit = np.arange(-282, 80, 1)
    yfit = interpolate.splev(xfit, tck, der=0)
    plt.plot(x, y, 'ro')
    plt.plot(xfit, yfit,'b')
    plt.plot(xfit, yfit)
    plt.title("Direct spline interpolation")
    plt.show()

def univariate():
    s = interpolate.InterpolatedUnivariateSpline(x, y)
```

```
    xfit = np.arange(0, 1.1, 0.0000001)
    yfit = s(xfit)
    plt.plot(x, y, 'ro')
    plt.plot(xfit, yfit,'green')
    plt.title("InterpolatedUnivariateSpline interpolation")
    plt.show()
```

5. Setting up the writing of the scripts for LibreCAD to create .dxf files

```
W = 7 #of outer ring
L = 10 #of outer ring
inner_W = 2 #of inner ring
inner_L = inner_W*L/W #of inner ring
t = 0.3 #thickness of outer ring
h = 0.001 #thickness of trace

def generate_command(xi, yi, width, height):
    outfile.write('li\n')
    outfile.write(f'{xi-width/2},{yi-height/2}\n')
    outfile.write(f'@{width},0\n')
    outfile.write(f'@0,{height}\n')
    outfile.write(f'@-{width},0\n')
    outfile.write('c\nk\n\n')

def generate_inner_ring(xi, yi, ss):
    global check_command_ran
    width = inner_W+(W+2*t-inner_W)*ss
    height = inner_L+(L+2*t-inner_L)*ss
    generate_command(xi, yi, width, height)
    check_command_ran = "inner_ring"

def generate_inner_ring_subtract(xi, yi, ss):
    global check_command_ran
    width = (inner_W-2*t)+(W-inner_W+2*t)*ss
    height = (inner_L-2*t)+(L-inner_L+2*t)*ss
    generate_command(xi, yi, width, height)
    check_command_ran = "inner_ring_subtract"

def generate_outer_ring(xi, yi):
    global check_command_ran
    width = W+2*t
    height = L+2*t
    generate_command(xi, yi, width, height)
    check_command_ran = "outer_ring"

def generate_outer_ring_subtract(xi, yi):
    global check_command_ran
    width = W
    height = L
    generate_command(xi, yi, width, height)
    check_command_ran = "outer_ring_subtract"

def generate_inner_rect(xi, yi):
    global check_command_ran
    width = inner_W
    height = inner_L
    generate_command(xi, yi, width, height)
    check_command_ran = "inner_rect"

def store_and_print(line: str, file):
    print(line)
    print(line, file=file)
```

6. Automation for generation of the .dxf files

```
def automation(xi, yi, ss):
    if file_name == "overall":
        generate_inner_ring(xi, yi, ss)
        generate_inner_ring_subtract(xi, yi, ss)
        generate_outer_ring(xi, yi)
        generate_outer_ring_subtract(xi, yi)
        generate_inner_rect(xi, yi)
    elif file_name == "inner_ring":
        generate_inner_ring(xi, yi, ss)
    elif file_name == "inner_ring_subtract":
        generate_inner_ring_subtract(xi, yi, ss)
    elif file_name == "outer_ring":
        generate_outer_ring(xi, yi)
    elif file_name == "outer_ring_subtract":
        generate_outer_ring_subtract(xi, yi)
    elif file_name == "inner_rect":
        generate_inner_rect(xi, yi)
```

7. Import needed libraries

```
import math
from re import L
from scipy import interpolate
import matplotlib.pyplot as plt
import numpy as np
```

8. Using the functions written above to read data of S11 curve, and interpolate it using direct interpolation (which gave the best result)

```
y, x = read_data()
x.reverse()
y.reverse()
print("unwrapped phase:", x)
print("unwrapped ss:", y)

#direct
tck = interpolate.splrep(x, y, s=0)
xfit = np.arange(0, 365, 1)
yfit = interpolate.splev(xfit, tck, der=0)
plt.plot(x, y, 'ro')
plt.plot(xfit, yfit,'b')
plt.plot(xfit, yfit)
plt.title("Direct spline interpolation --> ss over phase")
# plt.show()
```

9. Using the graph generated and functions above to get the phase at each point of the reflectarray and adjust the drawing accordingly

```
#actual code

#verifying setup

store_and_print("logs:\n", info_file)
store_and_print(f"frequency: {freq}Hz", info_file)
store_and_print(f"height of horn: {horn_height}mm", info_file)
store_and_print(f"number of cells horizontally: {across}", info_file)
store_and_print(f"number of cells vertically: {down}\n", info_file)

# accounting for first phase

first_phase_rad, origin_x , origin_y = get_phase(across//2, down//2)
first_phase_deg = first_phase_rad * 57.296 % 360
store_and_print(f"first phase, {first_phase_deg}, occurs at {origin_x, origin_y}", info_file)
```

```
# entire array

for x in range(across):
    for y in range(down):
        specific_phase, xi, yi = get_phase(x, y)
        specific_phase = (specific_phase - first_phase_rad) * 57.296 % 360
        ss = interpolate.splev(specific_phase, tck)
        printfile.write(f'At ({xi},{yi}), phase = {specific_phase}, ss = {ss}\n')
        automation(xi, yi, ss)

# verification

store_and_print(f"width: {across*UCsize_length_x}mm across", info_file)
store_and_print(f"length: {down*UCsize_length_y}mm down", info_file)
print(f"command ran: {check_command_ran}")
print(f"saving to file {file_name}\n")

info_file.close()
outfile.close()
printfile.close()
```

APPENDIX D: Code developed for calculation and plotting of spillover efficiency and illumination frequency of reflectarray

In this section, the code written in Python for calculations on the efficiency calculations (refer to Page 7 and 8) is presented, each with brief explanations on its purpose. There are 7 parts to this.

1. In a python file named machine.py, define the parameters of the feed horn element

```
#actual parameters
qe=qh=8.29

c = 3.0*10**8
GHz = 12.7
lamb = c/(GHz*10**9)

length_a = 0.283/lamb
length_b = 0.1835/lamb
```

2. Defining the functions according to the paper [6]. Power A denotes the power received by reflectarray elements and radiated by the feed. Power B is the power radiated by the feed that goes onto the array. Power C is the power radiated in the front forward facing hemisphere of the horn.

```
def fa(R):

    a = lambda y,x:
(((R/(R**2+x**2+y**2)**(1/2))**(qe+2))*((y**2)/(x**2+y**2))+((R/(R**2+x**2+y**2)**(1/2))**(qh+
1))*((x**2)/(x**2+y**2)))/((R**2+x**2+y**2)**(1/2))

    ans = 4*integrate.dblquad(a, 0, length_a/2, 0, length_b/2)[0]

    return ans

def fb(R):

    b = lambda y,x: R**(2*qe+1)*(R**2+x**2+y**2)**(-3/2-qe)

    ans = 4*integrate.dblquad(b, 0, length_a/2, 0, length_b/2)[0]

    return ans

def fc():
    return math.pi*(1/(1+2*qe)+1/(1+2*qh))
```

3. Import needed libraries

```
from scipy import integrate
import math
```

4. Defining illumination efficiency and spillover efficiency

```
fc = fc()

def Eff_ill(R):
    return fa(R)*fa(R)/(fb(R)*(length_a*length_b))

def Eff_spill(R):
```

```
    return fb(R)/fc

ei = Eff_ill(24)
es = Eff_spill(24)
```

5. In another file named plot.py, we imported the matplotlib package as well as the numpy package to generate a plot.

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

6. We imported machine.py to call the illumination efficiency and spillover efficiency calculation functions

```

from machine import Eff_ill, Eff_spill
```

7. Looping through different R values, where R is the height between the array and horn, we calculated the efficiency at each point to populate the graph graph.

```

starting_R = 1
ending_R = 20

efficiency=[]
es=[]
ei=[]
R=[]

steps = np.arange(starting_R, ending_R, 0.1)

for n in steps:
    R.append(n)
    efficiency_at_point = Eff_spill(n)*Eff_ill(n)
    es.append(Eff_spill(n))
    ei.append(Eff_ill(n))
    efficiency.append(efficiency_at_point)

plt.plot(R, efficiency, color='red', linestyle='-', label='μs×μi')
plt.plot(R, es, color='green', linestyle='--', label='μs')
plt.plot(R, ei, color='blue', linestyle='-.', label='μi')
plt.title('Graph of Efficiency against R/λ', fontsize=14)
plt.xlabel('R/λ', fontsize=14)
plt.ylabel('Efficiency', fontsize=14)
plt.legend()
plt.grid(True)
plt.show()
```