## SI Appendix 1

**Python Scripts Used to Generate DNA Staple Strand Sequences.** The following Python script was used to generate front monomer core oligos and head caps:

```
#!/usr/bin/env python

import sys
import string

###
# This program generates a six-helix bundle with bilateral pseudosymmetry
# 1. Create intermediate strands
# 2. Generate tokens
# 3. Generate oligos
###

# This function returns the complementary sequence
complement = string.maketrans('ACGTacgt','TGCAtgca')
def comp(s):
 return s.translate(complement)[::-1]

# This function returns the sequence with all unnecessary characters stripped out
def stripped_seq(raw_sequence):
  return ''.join([c for c in raw_sequence if c in string.letters])

# read in clonal strand sequence
input_file = file('p7308.txt', 'r')
CSS = stripped_seq(input_file.read())
CSS = CSS[-467:] + CSS[:-467]
input_file.close()

# Generate intermediate strands
num_LH_repeats = 13
num_RH_repeats = 14
start_marker = 0
fragment_ra = []
RH_overhang_length_ra = [26, 26, 40, 40, 2, 2]
LH_overhang_length_ra = [16, 16, 2, 2, 40, 40]
RC_length_ra = [20, -2, -2, 26, 26, 21]
LC_length_ra = [22,  44,  44, 16, 16, 21]

for RH_fragment_num in range(6):
        end_marker  = start_marker + RC_length_ra[RH_fragment_num]
        end_marker += num_RH_repeats*42
        end_marker += RH_overhang_length_ra[RH_fragment_num]
        fragment_ra.append(CSS[start_marker:end_marker])
        start_marker = end_marker

for LH_fragment_num in range(6):
        end_marker = start_marker + LC_length_ra[5 - LH_fragment_num]
        end_marker += num_LH_repeats*42
        end_marker += LH_overhang_length_ra[5 - LH_fragment_num]
        fragment_ra.append(CSS[start_marker:end_marker])
        start_marker = end_marker

I_strand_seq_ra = []
for I_strand_num in range(6):
        if I_strand_num%2 == 0:
                strand_seq = fragment_ra[11 - I_strand_num] + fragment_ra[I_strand_num]
                upstream_fragment   = strand_seq[LH_overhang_length_ra[I_strand_num]:]
                downstream_fragment = strand_seq[:LH_overhang_length_ra[I_strand_num]]
        else:
                strand_seq = fragment_ra[I_strand_num] + fragment_ra[11 - I_strand_num]
                upstream_fragment   = strand_seq[-LH_overhang_length_ra[I_strand_num]:]
                downstream_fragment = strand_seq[:-LH_overhang_length_ra[I_strand_num]]
        I_strand_seq_ra.append(upstream_fragment + downstream_fragment)

# Print intermediate strands
# output_file = file('WS_strands_6hb_v5.txt', 'w')
# for strand in I_strand_seq_ra:
#       output_file.write(strand + '\n')
#       sys.stdout.write(str(len(strand)) + '\n')
# output_file.close()

# Generate token array
token_ra = []
num_strand_tokens = len(I_strand_seq_ra[0])/7
for I_strand_num in range(6):
```

```
        start_marker = 0
        sub_token_ra = []
        for token_num in range(num_strand_tokens):
                end_marker = start_marker + 7

        sub_token_ra.append(comp(I_strand_seq_ra[I_strand_num][start_marker:end_marker]))
                start_marker = end_marker
        token_ra.append(sub_token_ra)

# Generate oligos
ITN_ra = [4, 4, 0, 2, 2, 0]                  # initial token number

oligo_ra = []
num_strand_oligos = num_strand_tokens/6
for I_strand_num in range(6):
        SO = int((I_strand_num%2 - 0.5)*(-2))
        TN = ITN_ra[I_strand_num]

        sub_oligo_ra = []
        for oligo_num in range(num_strand_oligos):
                oligo_seq  = token_ra[(I_strand_num + SO*2)%6][TN + 1]
                oligo_seq += token_ra[(I_strand_num + SO*2)%6][TN]
                oligo_seq += token_ra[(I_strand_num + SO)%6][num_strand_tokens - TN - 1]
                oligo_seq += token_ra[(I_strand_num + SO)%6][num_strand_tokens - TN - 2]
                oligo_seq += token_ra[I_strand_num][TN + 1]
                oligo_seq += token_ra[I_strand_num][TN]
                sub_oligo_ra.append(oligo_seq)
                TN += 6
        oligo_ra.append(sub_oligo_ra)

# Sort oligos by putting connecting oligos at the end of the list
sorted_oligo_ra = []
connector_oligo_ra = []
for I_strand_num in range(6):
        if I_strand_num%2 == 0:
                sorted_oligo_ra += oligo_ra[I_strand_num][:-1]
                connector_oligo_ra += oligo_ra[I_strand_num][-1:]
        else:
                connector_oligo_ra += oligo_ra[I_strand_num][:1]
                sorted_oligo_ra += oligo_ra[I_strand_num][1:]
#sorted_oligo_ra += connector_oligo_ra

# Generate head-cap oligos
head_cap_oligo_ra = []
head_cap_oligo_ra.append(comp(I_strand_seq_ra[1][:16]) + comp(I_strand_seq_ra[0][-16:]))
head_cap_oligo_ra.append(comp(I_strand_seq_ra[4][-40:]))
head_cap_oligo_ra.append(comp(I_strand_seq_ra[5][:40]))

# Generate tail-cap oligos
tail_cap_oligo_ra = []
tail_cap_oligo_ra.append(comp(I_strand_seq_ra[0][-42:-16]))
tail_cap_oligo_ra.append(comp(I_strand_seq_ra[1][16:42]))
tail_cap_oligo_ra.append(comp(I_strand_seq_ra[2][-42:-2]))
tail_cap_oligo_ra.append(comp(I_strand_seq_ra[3][2:42]))

#sorted_oligo_ra += head_cap_oligo_ra + tail_cap_oligo_ra
sorted_oligo_ra += head_cap_oligo_ra

output_file = file('front_monomer_oligos.txt', 'w')
num_oligos = 0
for oligo in sorted_oligo_ra:
        output_file.write(oligo + '\n')
        num_oligos += 1
output_file.close()
```

The following Python script was used to generate rear monomer core oligos and tail caps:

```
#!/usr/bin/env python

import sys
import string

###
# This program generates a six-helix bundle with bilateral pseudosymmetry
# 1. Create intermediate strands
# 2. Generate tokens
# 3. Generate oligos
```

```
###

# This function returns the complementary sequence
complement = string.maketrans('ACGTacgt','TGCAtgca')
def comp(s):
  return s.translate(complement)[::-1]

# This function returns the sequence with all unnecessary characters stripped out
def stripped_seq(raw_sequence):
  return ''.join([c for c in raw_sequence if c in string.letters])

# read in clonal strand sequence
input_file = file('p7308.txt', 'r')
CSS = stripped_seq(input_file.read())
CSS0 = CSS[-467:] + CSS[:-467]
CSS1 = CSS[-467 + 100:] + CSS[:-467 + 100]
CSS = ''
input_file.close()

# Generate intermediate strands
num_LH_repeats = 13
num_RH_repeats = 14
start_marker = 0
fragment_ra0 = []
fragment_ra1 = []
RH_overhang_length_ra = [26, 26, 40, 40, 2, 2]
LH_overhang_length_ra = [16, 16, 2, 2, 40, 40]
RC_length_ra = [20, -2, -2, 26, 26, 21]
LC_length_ra = [22,  44,  44, 16, 16, 21]

for RH_fragment_num in range(6):
        end_marker  = start_marker + RC_length_ra[RH_fragment_num]
        end_marker += num_RH_repeats*42
        end_marker += RH_overhang_length_ra[RH_fragment_num]
        fragment_ra0.append(CSS0[start_marker:end_marker])
        fragment_ra1.append(CSS1[start_marker:end_marker])
        start_marker = end_marker

for LH_fragment_num in range(6):
        end_marker = start_marker + LC_length_ra[5 - LH_fragment_num]
        end_marker += num_LH_repeats*42
        end_marker += LH_overhang_length_ra[5 - LH_fragment_num]
        fragment_ra0.append(CSS0[start_marker:end_marker])
        fragment_ra1.append(CSS1[start_marker:end_marker])
        start_marker = end_marker

strand_ra = []

for strand_num in range(6):
        if strand_num%2 == 0:
                strand0 = fragment_ra0[11 - strand_num] + fragment_ra0[strand_num]
                strand1 = fragment_ra1[11 - strand_num] + fragment_ra1[strand_num]
                strand_ra.append(strand0[-42 - RH_overhang_length_ra[strand_num]:] +
strand1[:42 + LH_overhang_length_ra[strand_num]])
        else:
                strand0 = fragment_ra0[strand_num] + fragment_ra0[11 - strand_num]
                strand1 = fragment_ra1[strand_num] + fragment_ra1[11 - strand_num]
                strand_ra.append(strand1[-42 - LH_overhang_length_ra[strand_num]:] +
strand0[:42 + RH_overhang_length_ra[strand_num]])

for strand in strand_ra:
        sys.stdout.write(strand + '\n')

token_ra = []
for strand_num in range(6):
        sub_token_ra = []
        num_tokens = 9
        for token_num in range(num_tokens):
                if strand_num%2 == 0:

        sub_token_ra.append(comp(strand_ra[strand_num][token_num*14:token_num*14 + 14]))
                else:
                        sub_token_ra.insert(0,
comp(strand_ra[strand_num][token_num*14:token_num*14 + 14]))
        token_ra.append(sub_token_ra)

oligo_ra = []
num_zones = 9
for zone_num in range(num_zones):
        oligo  = token_ra[(zone_num*2 - 1)%6][zone_num]
        oligo += token_ra[(zone_num*2 + 0)%6][zone_num]
```

```
        oligo += token_ra[(zone_num*2 + 1)%6][zone_num]
        oligo_ra.append(oligo)
        oligo  = token_ra[(zone_num*2 + 4)%6][zone_num]
        oligo += token_ra[(zone_num*2 + 3)%6][zone_num]
        oligo += token_ra[(zone_num*2 + 2)%6][zone_num]
        oligo_ra.append(oligo)


output_file = file('front_rear_connector_oligos.txt', 'w')
for oligo in oligo_ra[6:-6]:
        output_file.write(oligo + '\n')
output_file.close()


input_file = file('front_monomer_oligos.txt', 'r')
lines = input_file.readlines()
input_file.close()
v5_oligo_ra = [stripped_seq(line) for line in lines]

input_file = file('rear_monomer_oligos.txt', 'r')
lines = input_file.readlines()
input_file.close()
v6_oligo_ra = [stripped_seq(line) for line in lines]

for oligo_num in range(18):
        if v5_oligo_ra.count(oligo_ra[oligo_num]) == 1:
                sys.stdout.write('Oligo ' + str(oligo_num) + ' present in v5 oligo
array.\n')
        elif v6_oligo_ra.count(oligo_ra[oligo_num]) == 1:
                sys.stdout.write('Oligo ' + str(oligo_num) + ' present in v6 oligo
array.\n')
        else:
                sys.stdout.write(oligo_ra[oligo_num] + '\n')
```

The following Python script was used to generate front monomer tail connector oligos and rear head connector oligos:

```
#!/usr/bin/env python

import sys
import string

# This function returns the complementary sequence
complement = string.maketrans('ACGTacgt','TGCAtgca')
def comp(s):
  return s.translate(complement)[::-1]

# This function returns the sequence with all unnecessary characters stripped out
def stripped_seq(raw_sequence):
  return ''.join([c for c in raw_sequence if c in string.letters])

# read in clonal strand sequence
input_file = file('p7308.txt', 'r')
CSS = stripped_seq(input_file.read())
CSS0 = CSS[-467:] + CSS[:-467]
CSS1 = CSS[-467 + 100:] + CSS[:-467 + 100]
CSS = ''
input_file.close()

# Generate intermediate strands
num_LH_repeats = 13
num_RH_repeats = 14
start_marker = 0
fragment_ra0 = []
fragment_ra1 = []
RH_overhang_length_ra = [26, 26, 40, 40, 2, 2]
LH_overhang_length_ra = [16, 16, 2, 2, 40, 40]
RC_length_ra = [20, -2, -2, 26, 26, 21]
LC_length_ra = [22,  44,  44, 16, 16, 21]

for RH_fragment_num in range(6):
        end_marker  = start_marker + RC_length_ra[RH_fragment_num]
        end_marker += num_RH_repeats*42
        end_marker += RH_overhang_length_ra[RH_fragment_num]
```

```
        fragment_ra0.append(CSS0[start_marker:end_marker])
        fragment_ra1.append(CSS1[start_marker:end_marker])
        start_marker = end_marker

for LH_fragment_num in range(6):
        end_marker = start_marker + LC_length_ra[5 - LH_fragment_num]
        end_marker += num_LH_repeats*42
        end_marker += LH_overhang_length_ra[5 - LH_fragment_num]
        fragment_ra0.append(CSS0[start_marker:end_marker])
        fragment_ra1.append(CSS1[start_marker:end_marker])
        start_marker = end_marker

strand_ra = []

for strand_num in range(6):
        if strand_num%2 == 0:
                strand0 = fragment_ra0[11 - strand_num] + fragment_ra0[strand_num]
                strand1 = fragment_ra1[11 - strand_num] + fragment_ra1[strand_num]
                strand_ra.append(strand0[-42 - RH_overhang_length_ra[strand_num]:] +
strand1[:42 + LH_overhang_length_ra[strand_num]])
        else:
                strand0 = fragment_ra0[strand_num] + fragment_ra0[11 - strand_num]
                strand1 = fragment_ra1[strand_num] + fragment_ra1[11 - strand_num]
                strand_ra.append(strand1[-42 - LH_overhang_length_ra[strand_num]:] +
strand0[:42 + RH_overhang_length_ra[strand_num]])

for strand in strand_ra:
        sys.stdout.write(strand + '\n')

token_ra = []
for strand_num in range(6):
        sub_token_ra = []
        num_tokens = 9
        for token_num in range(num_tokens):
                if strand_num%2 == 0:

        sub_token_ra.append(comp(strand_ra[strand_num][token_num*14:token_num*14 + 14]))
                else:
                        sub_token_ra.insert(0,
comp(strand_ra[strand_num][token_num*14:token_num*14 + 14]))
        token_ra.append(sub_token_ra)


oligo_ra = []
num_zones = 9
for zone_num in range(num_zones):
        oligo  = token_ra[(zone_num*2 - 1)%6][zone_num]
        oligo += token_ra[(zone_num*2 + 0)%6][zone_num]
        oligo += token_ra[(zone_num*2 + 1)%6][zone_num]
        oligo_ra.append(oligo)
        oligo  = token_ra[(zone_num*2 + 4)%6][zone_num]
        oligo += token_ra[(zone_num*2 + 3)%6][zone_num]
        oligo += token_ra[(zone_num*2 + 2)%6][zone_num]
        oligo_ra.append(oligo)


output_file = file('front_rear_connector_oligos.txt', 'w')
for oligo in oligo_ra[6:-6]:
        output_file.write(oligo + '\n')
output_file.close()

input_file = file('front_monomer_oligos.txt', 'r')
lines = input_file.readlines()
input_file.close()
v5_oligo_ra = [stripped_seq(line) for line in lines]

input_file = file('rear_monomer_oligos.txt', 'r')
lines = input_file.readlines()
input_file.close()
v6_oligo_ra = [stripped_seq(line) for line in lines]

for oligo_num in range(18):
        if v5_oligo_ra.count(oligo_ra[oligo_num]) == 1:
                sys.stdout.write('Oligo ' + str(oligo_num) + ' present in v5 oligo
array.\n')
        elif v6_oligo_ra.count(oligo_ra[oligo_num]) == 1:
                sys.stdout.write('Oligo ' + str(oligo_num) + ' present in v6 oligo
array.\n')
        else:
                sys.stdout.write(oligo_ra[oligo_num] + '\n')
```