In [44]:
```
# Libraries
import numpy as np # Data manipulation
from pynq import allocate, Overlay, MMIO # Access HWH, BIT, and AXI-LITE
import matplotlib.pyplot as plt # Plot data
import time # Track elapsed time
import os # Access to OS interfaces dependent functionality
import scipy.io
import time
```

In [45]:
```
# Explicitly set working directory
os.chdir("/home/xilinx/jupyter_notebooks/Notebook_Tests/stft_test")
print(f"Verifying notebook's working directory: {os.getcwd()}")
print(os.listdir())
```

```
Verifying notebook's working directory: /home/xilinx/jupyter_notebooks/Noteb
ook_Tests/stft_test
['design_stft.bit', 'design_stft.hwh', 'PLUTO_TEST_DATA.mat', 'stft_test.ipy
nb', '.ipynb_checkpoints', 'stft_test.pdf']
```

In [46]:
```
# Explicitly set working directory
os.chdir("/home/xilinx/jupyter_notebooks/Notebook_Tests/stft_test")
print(f"Verifying notebook's working directory: {os.getcwd()}")
print(f"Files in current working directory: {os.listdir()}")

# Download bistream & parse HWH for IPI block diagram
ol = Overlay("design_stft.bit") # hwh is parsed here

# Interrogate HWH & display information about design
print(f"IP blocks: {ol.ip_dict.keys()}")
```

```
Verifying notebook's working directory: /home/xilinx/jupyter_notebooks/Noteb
ook_Tests/stft_test
Files in current working directory: ['design_stft.bit', 'design_stft.hwh',
'PLUTO_TEST_DATA.mat', 'stft_test.ipynb', '.ipynb_checkpoints', 'stft_test.p
df']
IP blocks: dict_keys(['axi_dma_0', 'zynq_ultra_ps_e_0'])
```

In [58]:
```
# Load .m file
data_pluto = scipy.io.loadmat('PLUTO_TEST_DATA.mat')

# Extract the data_samples
data_samples = data_pluto['data_samples'].flatten()
print(f"type(data_samples): {type(data_samples)}")
data_sample = data_samples[0:2047]
print(f"type(data_sample): {type(data_sample)}")
print(f"len(data_sample): {len(data_sample)}")
print("Data samples:")
for idx in range(5):
    print(f"     data_sample[{idx}]: {data_sample[idx]}")
```

```
type(data_samples): <class 'numpy.ndarray'>
type(data_sample): <class 'numpy.ndarray'>
len(data_sample): 2047
Data samples:
     data_sample[0]: -0.8944271909999159j
     data_sample[1]: 0j
     data_sample[2]: (0.35355339059327373-0.35355339059327373j)
     data_sample[3]: (-0.35355339059327373+0.7071067811865475j)
     data_sample[4]: (0.48507125007266594+0j)
```

In [59]:
```
# Set up IO buffers to send and recieve data from DMA through STFT
input_buffer = allocate(shape=(len(data_sample),), dtype=np.uint32)
output_buffer = allocate(shape=(len(data_sample),), dtype=np.uint32)
time.sleep(2)
```

In [60]:
```
# Imaginary in upper bits
for idx in range(len(data_sample)):
    real = int(np.real(data_sample[idx]) * 32767)
    imag = int(np.imag(data_sample[idx]) * 32767)
    # Cast to int16 to handle wrap-around, then to uint32 for packing
    real16 = np.uint32(np.int16(real)) # Ensure 2s complement behaviour
    imag16 = np.uint32(np.int16(imag))
    packed = (imag16 << 16) | (real16 & 0xFFFF)
    input_buffer[idx] = packed
    if idx == 3:
        print(f"data_sample[idx]: {data_sample[idx]}")
        print(f"real: {real}")
        print(f"imag: {imag}")
        print(f"real16: {real16}")
        print(f"imag16: {imag16}")
        print(f"packed: {packed}")
```

```
data_sample[idx]: (-0.35355339059327373+0.7071067811865475j)
real: -11584
imag: 23169
real16: 4294955712
imag16: 23169
packed: 1518457536
```

In [61]:
```
dma = ol.axi_dma_0
# Perform exchange of data w/ DMA, as s_axis & m_axis of IP core is conn to DM
dma.sendchannel.transfer(input_buffer)
dma.recvchannel.transfer(output_buffer)
dma.sendchannel.wait()
dma.recvchannel.wait()
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Input In [61], in <cell line: 6>()
      4 dma.recvchannel.transfer(output_buffer)
      5 dma.sendchannel.wait()
----> 6 dma.recvchannel.wait()

File /usr/local/share/pynq-venv/lib/python3.10/site-packages/pynq/lib/dma.p
y:171, in _SDMAChannel.wait(self)
    169     raise RuntimeError("DMA channel not started")
    170 while True:
--> 171     error = self._mmio.read(self._offset + 4)
    172     if self.error:
    173         if error & 0x10:

File /usr/local/share/pynq-venv/lib/python3.10/site-packages/pynq/mmio.py:8
2, in MMIO.read(self, offset, length, word_order)
     79     else:
     80         raise ValueError("Device does not have capabilities for MMI
O")
---> 82 def read(self, offset=0, length=4, word_order="little"):
     83     """The method to read data from MMIO.
     84
     85     For the `word_order` parameter, it is only effective when
   (...)
    106
    107     """
    108     if length not in [1, 2, 4, 8]:

KeyboardInterrupt:
```

In [63]:
```
result = output_buffer.astype(np.int16)
print(f"len(result): {len(result)}")
print(f"result: {result}")
```

```
len(result): 2047
result: [0 0 0 ... 0 0 0]
```

In [ ]:

In [ ]:

In [ ]:

In [4]:
```
dma = ol.axi_dma_0

print(f"\ndma: {ol.ip_dict['axi_dma_0']}")




# Get name of ram_re
ram_re = ol.ip_dict['ram_re_0']['fullpath']
print(f"\nram_re name: {ram_re}")

# Get base address of ram_re
print(f"\nram_re address: {int(hex(ol.ip_dict['ram_re_0']['phys_addr']),16)}")
print(f"ram_re address hex: {hex(ol.ip_dict['ram_re_0']['phys_addr'])}")
ram_re_addr = 0xa0010000

# Get address range of ram_re
print(f"\nram_re address range: {int(hex(ol.ip_dict['ram_re_0']['addr_range'])
print(f"ram_re address range hex: {hex(ol.ip_dict['ram_re_0']['addr_range'])}"
ram_re_addr_range = 0x10000

# Create MMIO object to interact with AXI Lite interface of ram_re
ram_re_mmio = MMIO(ram_re_addr, ram_re_addr_range)
print(f"\nram_re_mmio: {ram_re_mmio}")
```

```
ram_re name: ram_re_0

ram_re address: 2684420096
ram_re address hex: 0xa0010000

ram_re address range: 65536
ram_re address range hex: 0x10000

ram_re_mmio: <pynq.mmio.MMIO object at 0xffff7ad508e0>
```

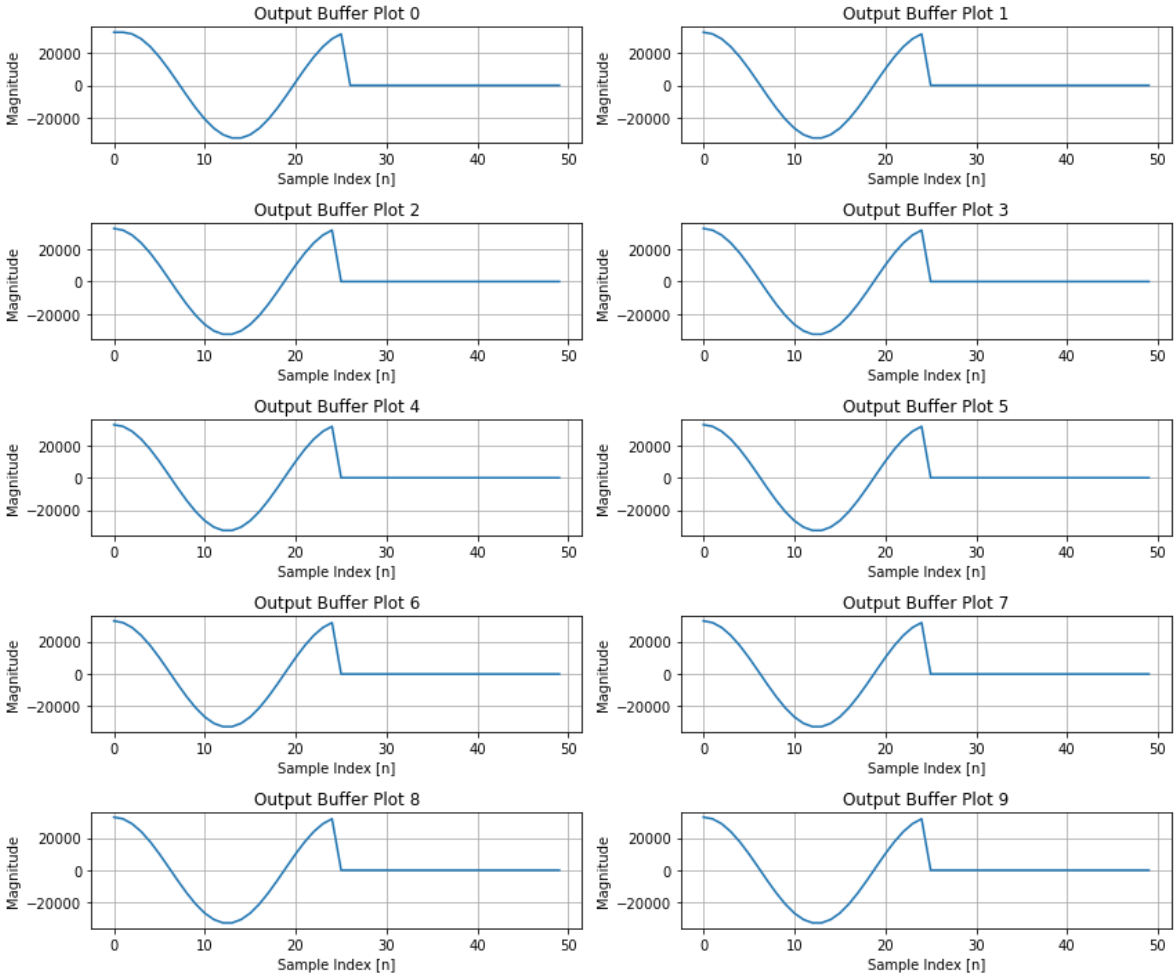
In [5]:
```
'''
#print("===== AXI-LITE =====\n\n")
input_data = 1
address_offset = 0x0 # Must be multiple of 4
ram_re_mmio.write(address_offset, input_data)
time.sleep(0.1)
print(f"Data sent to ram_re: {ram_re_mmio.read(address_offset)}")

#print("\n\n===== DMA =====\n\n")
# Get the DMA instance
dma = ol.axi_dma

plt.subplots(figsize=(12,10))
for i in range(10):
    if i == 7:
        input_data = 0
        address_offset = 0x0 # Must be multiple of 4
        ram_re_mmio.write(address_offset, input_data)
        time.sleep(0.1)
        print(f"Data sent to ram_re: {ram_re_mmio.read(address_offset)} after

    # Allocate memory for DMA transfer
    output_buffer = allocate(shape=(50,), dtype=np.uint16)

    # Start DMA transfer
    dma.recvchannel.transfer(output_buffer)
    dma.recvchannel.wait()

    # Adapt DMA output
    result = output_buffer.astype(np.int16)

    if i == 1:
        temp_output_buffer_1 = output_buffer
        temp_result_1 = result
    if i == 8:
        temp_output_buffer_8 = output_buffer
        temp_result_8 = result

    # Plot data
    plt.subplot(5,2,i+1)
    plt.plot(result, label="dsp_RAM")
    plt.xlabel("Sample Index [n]")
    plt.ylabel("Magnitude")
    plt.title(f"Output Buffer Plot {i}")
    plt.grid(True)
plt.tight_layout()
plt.show()

#print(f"DMA output of iter {i}:\n {output_buffer}")
print(f"\ndma_re data of iter 1:\n {temp_result_1}")

rint(f"\ndma_re data of iter 8:\n {temp_result_8}")
'''
```

Out[5]:
```
'\n#print("===== AXI-LITE =====\n\n")\ninput_data = 1\naddress_offset = 0x0
# Must be multiple of 4\nram_re_mmio.write(address_offset, input_data)\ntim
e.sleep(0.1)\nprint(f"Data sent to ram_re: {ram_re_mmio.read(address_offse
t)}")\n\n#print("\n\n===== DMA =====\n\n")\n# Get the DMA instance\ndma = o
l.axi_dma \n\nplt.subplots(figsize=(12,10))\nfor i in range(10):\n    if i =
= 7:\n        input_data = 0\n        address_offset = 0x0 # Must be multipl
e of 4\n        ram_re_mmio.write(address_offset, input_data)\n        time.
sleep(0.1)\n        print(f"Data sent to ram_re: {ram_re_mmio.read(address_o
ffset)} after iter {i}")\n\n    # Allocate memory for DMA transfer\n    outp
ut_buffer = allocate(shape=(50,), dtype=np.uint16)\n\n    # Start DMA transf
er\n    dma.recvchannel.transfer(output_buffer)\n    dma.recvchannel.wait()
\n\n    # Adapt DMA output\n    result = output_buffer.astype(np.int16)\n
\n    if i == 1:\n        temp_output_buffer_1 = output_buffer\n        temp
_result_1 = result\n    if i == 8:\n        temp_output_buffer_8 = output_bu
ffer\n        temp_result_8 = result\n        \n    # Plot data\n    plt.sub
plot(5,2,i+1)\n    plt.plot(result, label="dsp_RAM")\n    plt.xlabel("Sample
Index [n]")\n    plt.ylabel("Magnitude")\n    plt.title(f"Output Buffer Plot
{i}")\n    plt.grid(True)\nplt.tight_layout()    \nplt.show()\n\n#print(f"DM
A output of iter {i}:\n {output_buffer}")\nprint(f"\ndma_re data of iter
1:\n {temp_result_1}")\n\nrint(f"\ndma_re data of iter 8:\n {temp_result_
8}")\n'
```

Test 2

In [6]:
```
#print("===== AXI-LITE =====\n\n")
input_data = 0 # Reset signal
address_offset = 0x0 # Must be multiple of 4
ram_re_mmio.write(address_offset, input_data)
time.sleep(0.1)
print(f"Reset signal sent to ram_re: {ram_re_mmio.read(address_offset)}")
input_data = 1 # Enable signal
ram_re_mmio.write(address_offset, input_data)
time.sleep(0.1)
print(f"Data sent to ram_re: {ram_re_mmio.read(address_offset)}")

#print("\n\n===== DMA =====\n\n")
# Get the DMA instance
dma = ol.axi_dma

plt.subplots(figsize=(12,10))
for i in range(10):
    if i == 7:
        input_data = 0
        address_offset = 0x0 # Must be multiple of 4
        ram_re_mmio.write(address_offset, input_data)
        time.sleep(0.1)
        print(f"Data sent to ram_re: {ram_re_mmio.read(address_offset)} after

    # Allocate memory for DMA transfer
    output_buffer = allocate(shape=(50,), dtype=np.uint16)
    output_buffer.fill(0) # Ensure buffer is zeroed before transfer - flush

    # Start DMA transfer
    dma.recvchannel.transfer(output_buffer)
    dma.recvchannel.wait()

    # Adapt DMA output
    result = output_buffer.astype(np.int16)

    if i == 1:
        temp_output_buffer_1 = output_buffer
        temp_result_1 = result
    if i == 8:
        temp_output_buffer_8 = output_buffer
        temp_result_8 = result

    # Plot data
    plt.subplot(5,2,i+1)
    plt.plot(result, label="dsp_RAM")
    plt.xlabel("Sample Index [n]")
    plt.ylabel("Magnitude")
    plt.title(f"Output Buffer Plot {i}")
    plt.grid(True)
plt.tight_layout()
plt.show()

#print(f"DMA output of iter {i}:\n {output_buffer}")
print(f"\ndma_re data of iter 1:\n {temp_result_1}")
print(f"\ndma_re data of iter 8:\n {temp_result_8}")
```

```
Reset signal sent to ram_re: 0
Data sent to ram_re: 1
Data sent to ram_re: 0 after iter 7
```




```
dma_re data of iter 1:
 [ 32767  31738  28714  23886  17558  10126   2058  -6140 -13952 -20888
 -26510 -30466 -32510 -32510 -30466 -26510 -20888 -13952  -6140   2058
  10126  17558  23886  28714  31738      0      0      0      0      0
      0      0      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0      0      0]

dma_re data of iter 8:
 [ 32767  31738  28714  23886  17558  10126   2058  -6140 -13952 -20888
 -26510 -30466 -32510 -32510 -30466 -26510 -20888 -13952  -6140   2058
  10126  17558  23886  28714  31738      0      0      0      0      0
      0      0      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0      0      0]
```

In [ ]:

In [ ]: