In [1]:
```
# 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
```

In [2]:
```
# Explicitly set working directory
os.chdir("/home/xilinx/jupyter_notebooks/Notebook_Tests/dsp_test")
print(f"Verifying notebook's working directory: {os.getcwd()}")
print(os.listdir())
```

```
Verifying notebook's working directory: /home/xilinx/jupyter_notebooks/Notebook_Te
sts/dsp_test
['design_dsp.bit', 'dsp_test.ipynb', '.ipynb_checkpoints', 'design_dsp.hwh']
```

In [3]:
```
# Explicitly set working directory
os.chdir("/home/xilinx/jupyter_notebooks/Notebook_Tests/dsp_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_dsp.bit") # hwh is parsed here

# Interrogate HWH & display information about design
print(f"IP blocks: {ol.ip_dict.keys()}")
print(f"\nram_re: {ol.ip_dict['ram_re_0']}")
```

```
Verifying notebook's working directory: /home/xilinx/jupyter_notebooks/Notebook_Te
sts/dsp_test
Files in current working directory: ['design_dsp.bit', 'dsp_test.ipynb', '.ipynb_c
heckpoints', 'design_dsp.hwh']


IP blocks: dict_keys(['axi_dma', 'ram_re_0', 'zynq_ultra_ps_e_0'])

ram_re: {'type': 'CoRSoC:VMC:ram_re:1.0', 'mem_id': 'ram_re_ip_s_axi', 'memtype':
'REGISTER', 'gpio': {}, 'interrupts': {}, 'parameters': {'Component_Name': 'design
_1_ram_re_0_2', 'EDK_IPTYPE': 'PERIPHERAL', 'C_RAM_RE_IP_S_AXI_BASEADDR': '0xA0010
000', 'C_RAM_RE_IP_S_AXI_HIGHADDR': '0xA001FFFF', 'CLK_DOMAIN': 'design_1_zynq_ult
ra_ps_e_0_0_pl_clk0', 'FREQ_HZ': '96968727', 'HAS_TKEEP': '0', 'HAS_TLAST': '1',
'HAS_TREADY': '0', 'HAS_TSTRB': '0', 'INSERT_VIP': '0', 'LAYERED_METADATA': 'xilin
x.com:interface:datatypes:1.0 {TDATA {datatype {name {attribs {resolve_type immedi
ate dependency {} format string minimum {} maximum {}} value {}} bitwidth {attribs
{resolve_type immediate dependency {} format long minimum {} maximum {}} value 16}
bitoffset {attribs {resolve_type immediate dependency {} format long minimum {} ma
ximum {}} value 0} real {fixed {fractwidth {attribs {resolve_type immediate depend
ency {} format long minimum {} maximum {}} value 15} signed {attribs {resolve_type
immediate dependency {} format bool minimum {} maximum {}} value true}}}}}}', 'PHA
SE': '0.0', 'TDATA_NUM_BYTES': '2', 'TDEST_WIDTH': '0', 'TID_WIDTH': '0', 'TUSER_W
IDTH': '0', 'ADDR_WIDTH': '1', 'ARUSER_WIDTH': '0', 'AWUSER_WIDTH': '0', 'BUSER_WI
DTH': '0', 'DATA_WIDTH': '32', 'HAS_BRESP': '1', 'HAS_BURST': '0', 'HAS_CACHE':
'0', 'HAS_LOCK': '0', 'HAS_PROT': '0', 'HAS_QOS': '0', 'HAS_REGION': '0', 'HAS_RRE
SP': '1', 'HAS_WSTRB': '1', 'ID_WIDTH': '0', 'MAX_BURST_LENGTH': '1', 'NUM_READ_OU
TSTANDING': '1', 'NUM_READ_THREADS': '1', 'NUM_WRITE_OUTSTANDING': '1', 'NUM_WRITE
_THREADS': '1', 'PROTOCOL': 'AXI4LITE', 'READ_WRITE_MODE': 'READ_WRITE', 'RUSER_BI
TS_PER_BYTE': '0', 'RUSER_WIDTH': '0', 'SUPPORTS_NARROW_BURST': '0', 'WUSER_BITS_P
ER_BYTE': '0', 'WUSER_WIDTH': '0'}, 'registers': {}, 'driver': <class 'pynq.overla
y.DefaultIP'>, 'device': <pynq.pl_server.embedded_device.EmbeddedDevice object at
0xffff8bd5c6a0>, 'state': None, 'bdtype': None, 'phys_addr': 2684420096, 'addr_ran
ge': 65536, 'fullpath': 'ram_re_0'}
```

In [4]:
```
# 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']),16)}
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 0xffff8bd5c970>
```

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 iter

    # 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)\ntime.sleep(0.1)\n
print(f"Data sent to ram_re: {ram_re_mmio.read(address_offset)}")\n\n#print("\n\n=
==== DMA =====\n\n")\n# Get the DMA instance\ndma = ol.axi_dma \n\nplt.subplots(fi
gsize=(12,10))\nfor i in range(10):\n    if i == 7:\n        input_data = 0\n
address_offset = 0x0 # Must be multiple of 4\n        ram_re_mmio.write(address_of
fset, input_data)\n        time.sleep(0.1)\n        print(f"Data sent to ram_re:
{ram_re_mmio.read(address_offset)} after iter {i}")\n\n    # Allocate memory for D
MA transfer\n    output_buffer = allocate(shape=(50,), dtype=np.uint16)\n\n    # S
tart DMA transfer\n    dma.recvchannel.transfer(output_buffer)\n    dma.recvchanne
l.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_resul
t_1 = result\n    if i == 8:\n        temp_output_buffer_8 = output_buffer\n
temp_result_8 = result\n        \n    # Plot data\n    plt.subplot(5,2,i+1)\n    p
lt.plot(result, label="dsp_RAM")\n    plt.xlabel("Sample Index [n]")\n    plt.ylab
el("Magnitude")\n    plt.title(f"Output Buffer Plot {i}")\n    plt.grid(True)\npl
t.tight_layout()    \nplt.show()\n\n#print(f"DMA output of iter {i}:\n {output_buf
fer}")\nprint(f"\ndma_re data of iter 1:\n {temp_result_1}")\n\nrint(f"\ndma_re da
ta of iter 8:\n {temp_result_8}")\n'
```

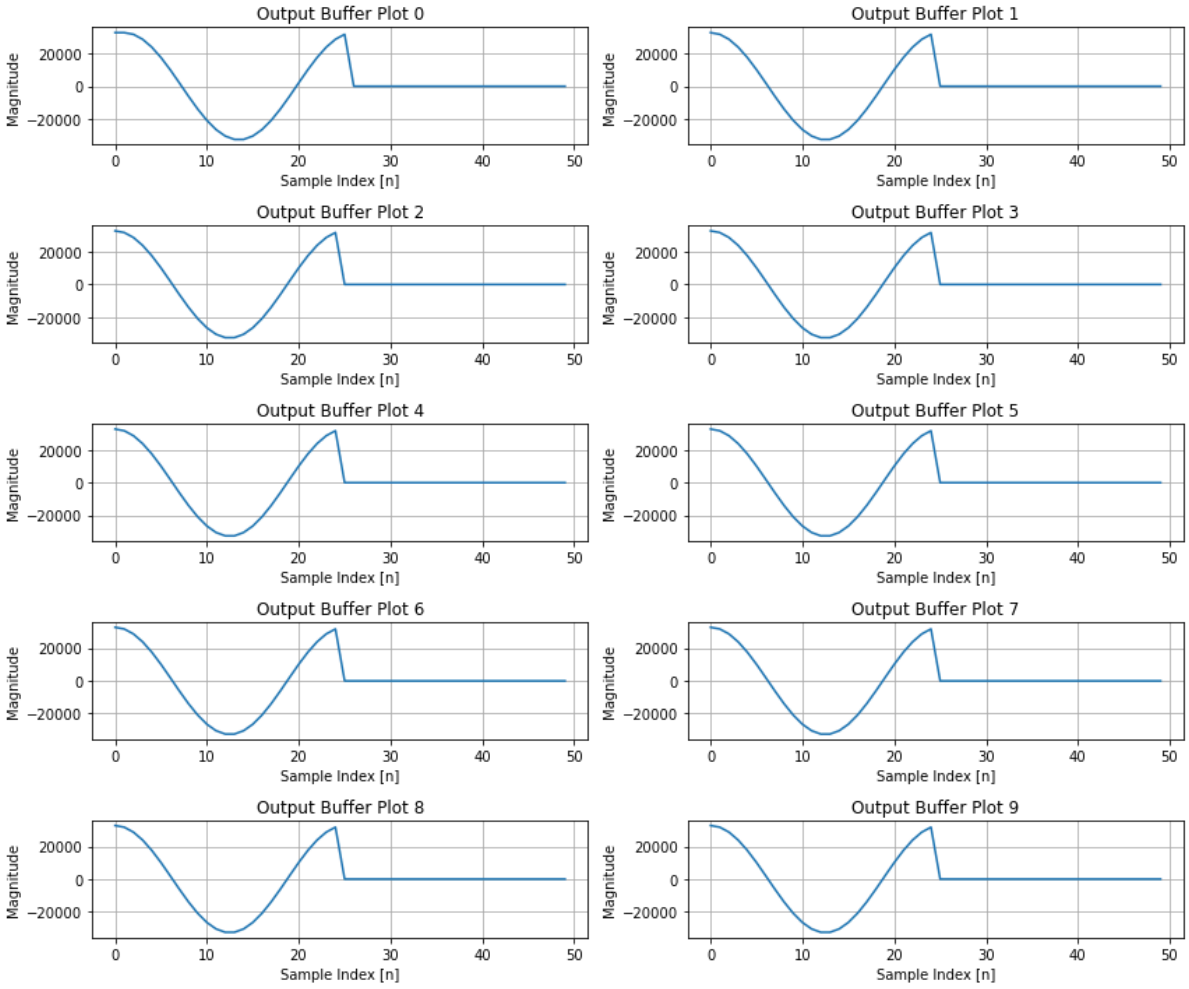
Test 2

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 iter
```

```
    # 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 [ ]: