FPGA Developers’ Forum (FDF) meeting 2024
11-13 Jun 2024 - CERN

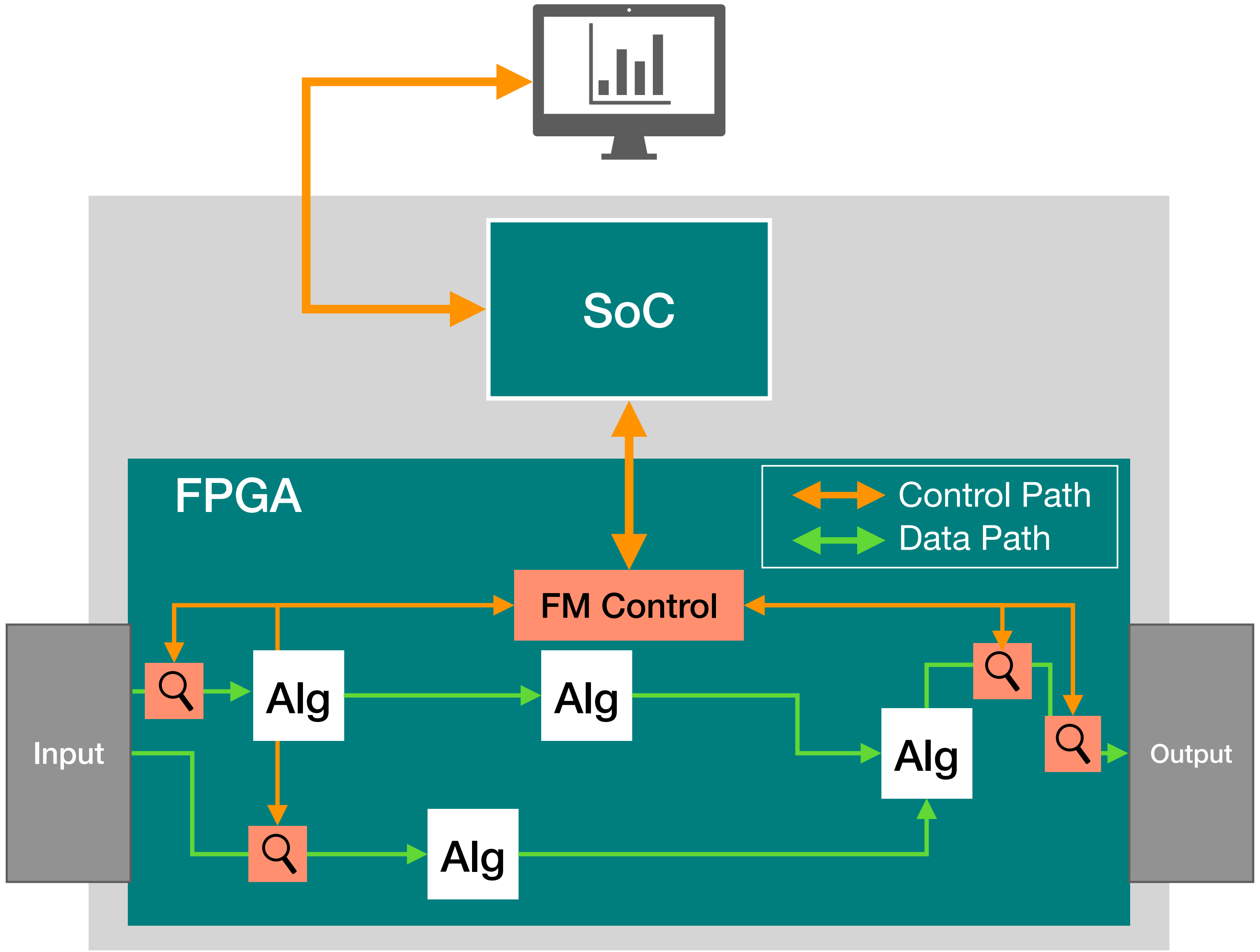
Fast Monitoring of FPGA algorithms using SpyBuffers

Iacopo Longarini, Priya Sundararajan (UCI)
Benjamin J. Rosser (U. Chicago)

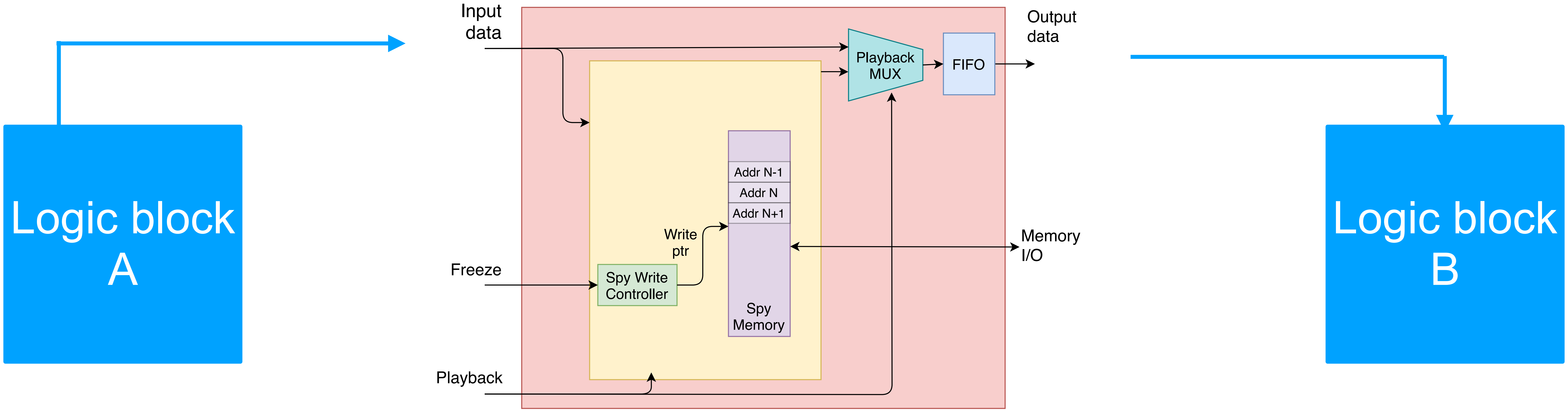
THE UNIVERSITY OF CHICAGO
UCI University of California, Irvine

Fast monitoring: introduction

- **Fast Monitoring (FM)** helps in performing real-time debugging of a FPGA firmware, as well in providing useful information when errors are detected and providing quality control during operation
- **The basic building block is the SpyBuffer (SB)** 🔍
  - SpyBuffers can be inserted at the input / output of several firmware blocks in order to “Spy” (monitor) the **data path**
  - Monitoring data be accessed thanks to an interface or a service running on a SoC via the **control path**
  - Data can be also injected in the firmware from a SpyBuffer, enabling advanced debugging features, reproduction of issues, etc
  - SB are controlled by a specific FM control block that needs to be implemented in the firmware
- SpyBuffer implementation in Verilog and software resources: **https://gitlab.cern.ch/spybuffer/**



SpyBuffer: in detail

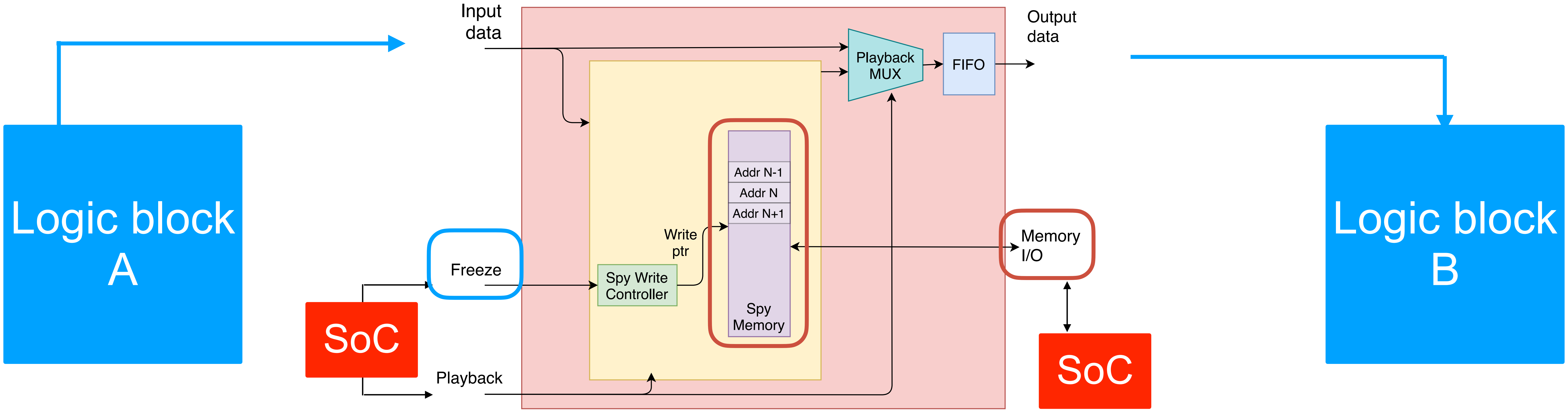


- **SpyBuffers** are small firmware blocks sitting between two logic blocks, allowing to “Spy” and “Inject” bitwords in the connection between them

- A copy of the all valid incoming data is stored to a circular buffer and passed to the second firmware block without added latency

- Optionally, a FIFO can be added (clock domain transition, buffered interfaces)

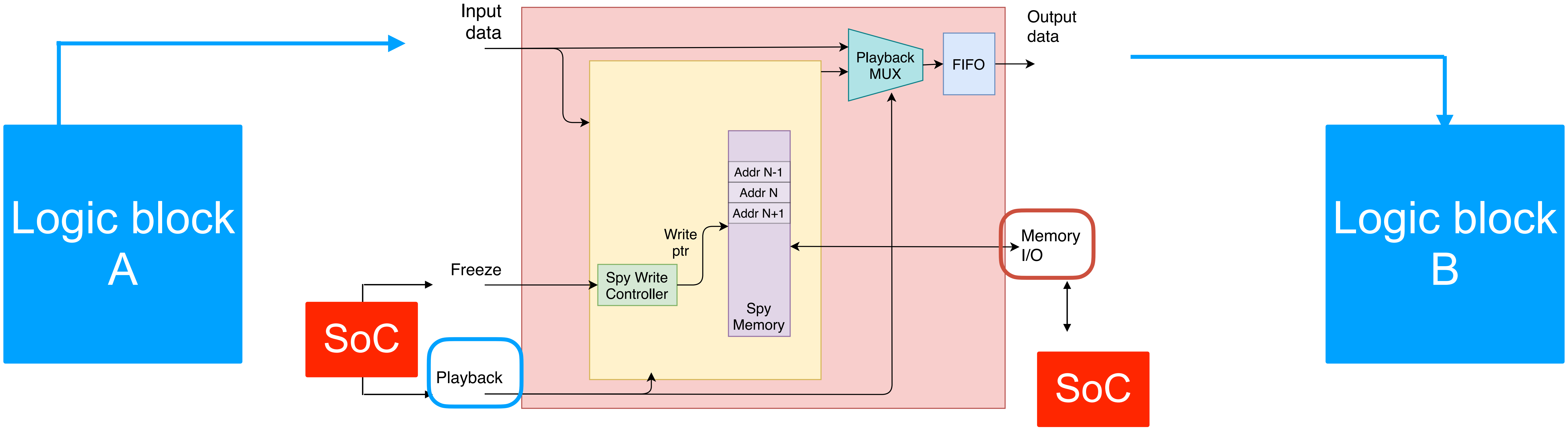
SpyBuffer: Memory I/O & Freeze mode



- SB memory content can be read (e.g. from a script running on a SoC)
  →This allows a simple way to get a “snapshot” of the data transmitted between A and B

- When the “**Freeze**” signal is asserted from the SoC:
  - The transmission of data from A to B is not interrupted
  - Monitoring is stopped: incoming words from A are not written to the SpyMemory

SpyBuffer: Playback mode

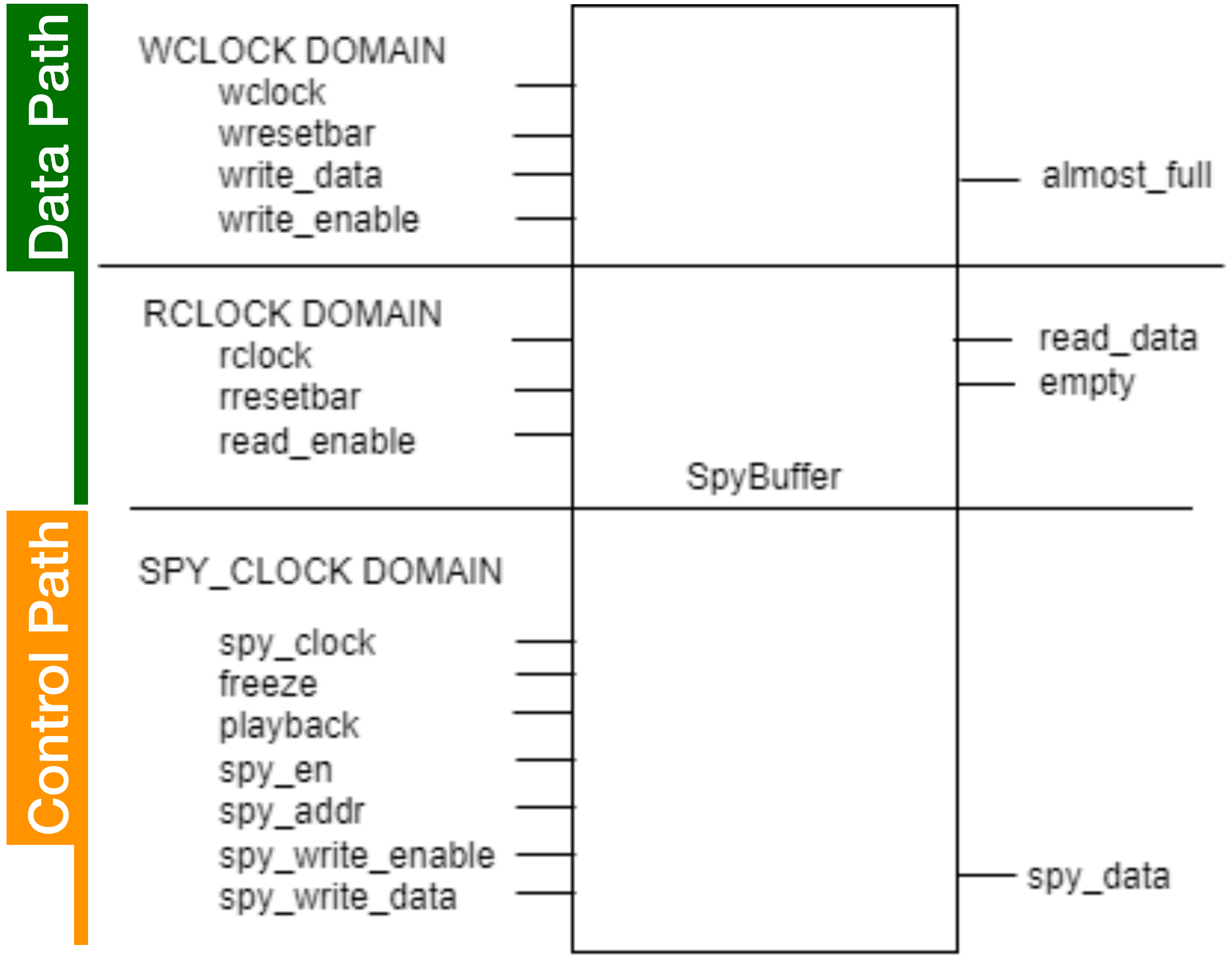


- With a SB in frozen state, it is possible to write test words (e.g. from simulation) to the circular buffer
- **Then, by asserting the Playback signal:**
  - MUX will replace the incoming data from A with the content of the circular buffer
  - Content of the SpyMemory will be sent to B
  - Optionally the playback can be set to a “loop” mode, where the content of the memory is continuously sent to B

SpyBuffer: firmware block interface & configuration

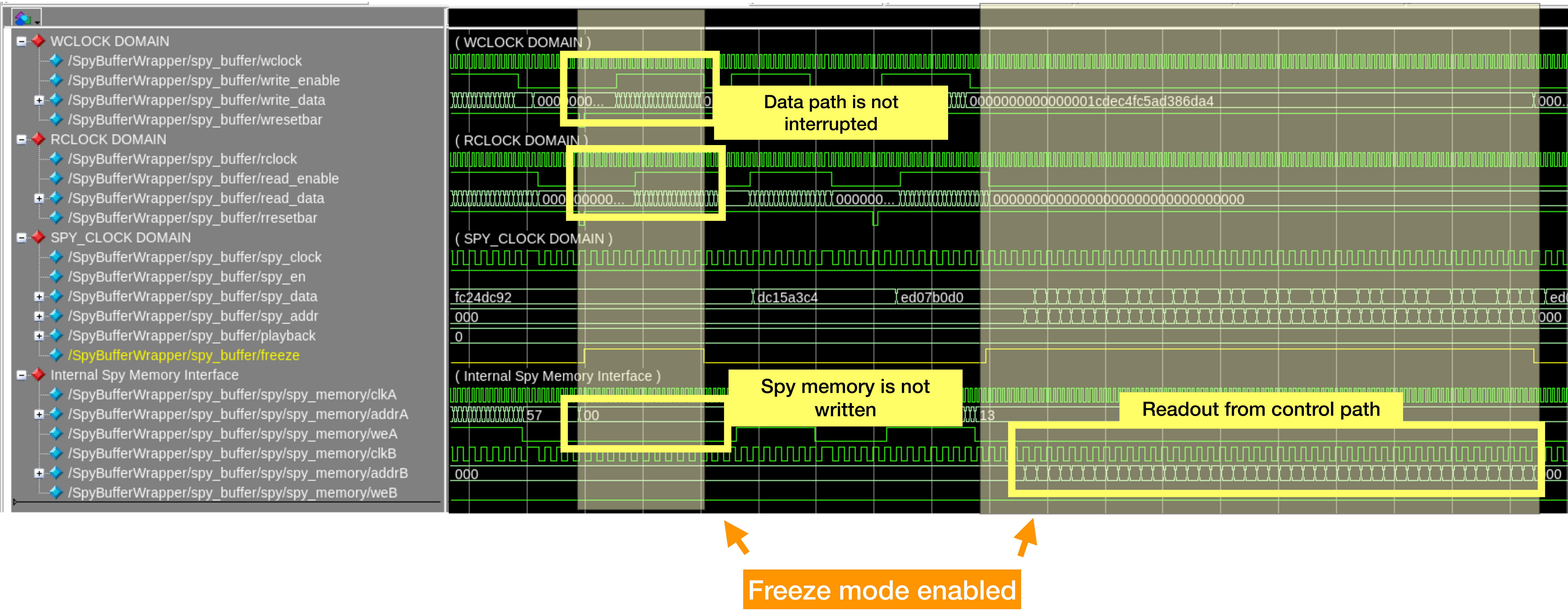
- SpyBuffer will support up to three clock domains, in case a dual-clock FIFO is added in the data path
- Control path connections include memory I/O and controllers for Freeze and Playback modes

Verilog Parameter	Description
DATA_WIDTH_A	Width of write_data, read_data (data path)
DATA_WIDTH_B	Width of spy_data (control path)
SPY_MEM_WIDTH_A	Set internal address width of SpyMemory to store incoming data
SPY_MEM_WIDTH_B	Width of spy_addr. This is fixed and is determined by the control interface
PASSTHROUGH	1: no FIFO, wclock and rclock are identical 0: FIFO inserted to handle wclock, rclock domain crossing



Waveforms - Freeze and readout

- Waveforms obtained from cocotb simulation



WCLOCK DOMAIN
/SpyBufferWrapper/spy_buffer/wclock
/SpyBufferWrapper/spy_buffer/write_enable
/SpyBufferWrapper/spy_buffer/write_data
/SpyBufferWrapper/spy_buffer/wresetbar
RCLOCK DOMAIN
/SpyBufferWrapper/spy_buffer/rclock
/SpyBufferWrapper/spy_buffer/read_enable
/SpyBufferWrapper/spy_buffer/read_data
/SpyBufferWrapper/spy_buffer/rresetbar
SPY_CLOCK DOMAIN
/SpyBufferWrapper/spy_buffer/spy_clock
/SpyBufferWrapper/spy_buffer/spy_en
/SpyBufferWrapper/spy_buffer/spy_data
/SpyBufferWrapper/spy_buffer/spy_addr
/SpyBufferWrapper/spy_buffer/playback
/SpyBufferWrapper/spy_buffer/freeze
Internal Spy Memory Interface
/SpyBufferWrapper/spy_buffer/spy/spy_memory/clkA
/SpyBufferWrapper/spy_buffer/spy/spy_memory/addrA
/SpyBufferWrapper/spy_buffer/spy/spy_memory/weA
/SpyBufferWrapper/spy_buffer/spy/spy_memory/clkB
/SpyBufferWrapper/spy_buffer/spy/spy_memory/addrB
/SpyBufferWrapper/spy_buffer/spy/spy_memory/weB

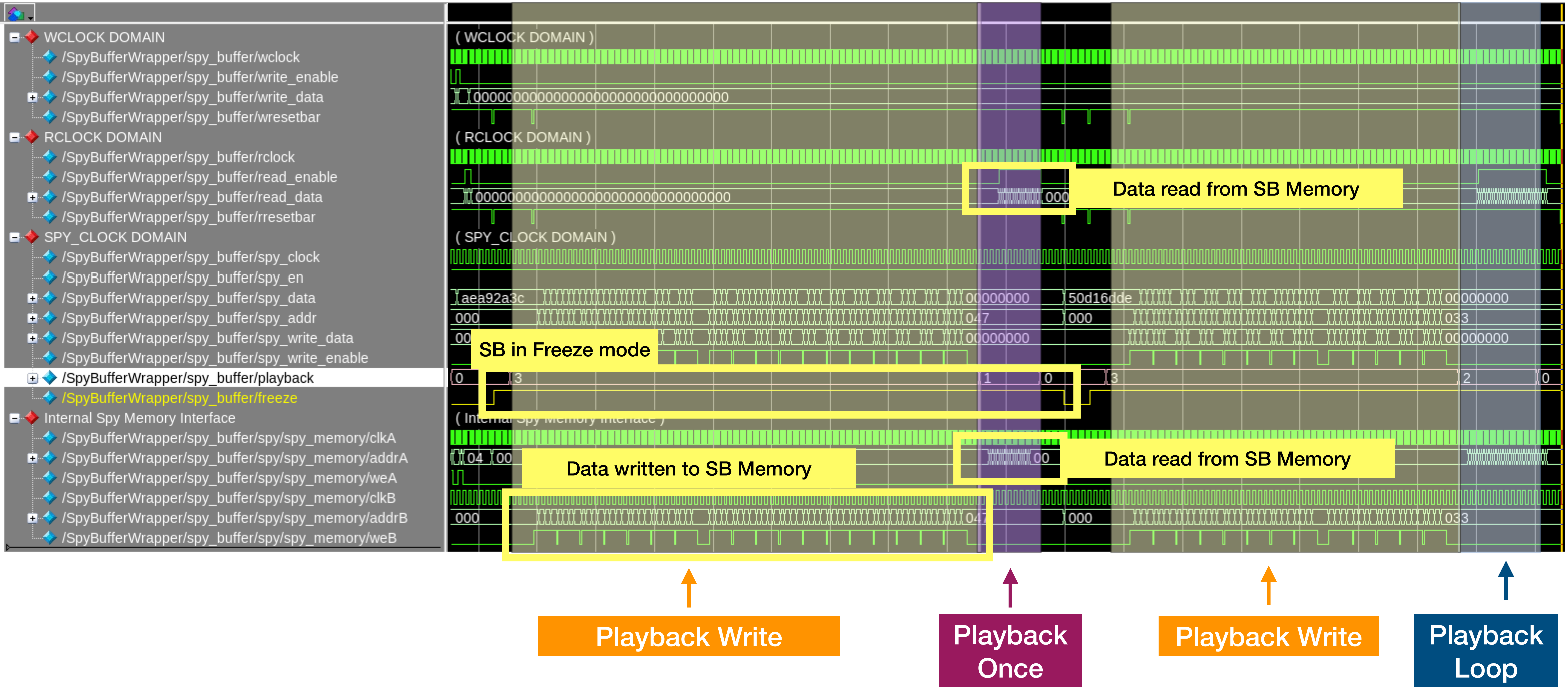
Data path is not interrupted

Spy memory is not written

Readout from control path

Freeze mode enabled

Waveform - Playback



WCLOCK DOMAIN
/SpyBufferWrapper/spy_buffer/wclock
/SpyBufferWrapper/spy_buffer/write_enable
/SpyBufferWrapper/spy_buffer/write_data
/SpyBufferWrapper/spy_buffer/wresetbar
RCLOCK DOMAIN
/SpyBufferWrapper/spy_buffer/rclock
/SpyBufferWrapper/spy_buffer/read_enable
/SpyBufferWrapper/spy_buffer/read_data
/SpyBufferWrapper/spy_buffer/rresetbar
SPY_CLOCK DOMAIN
/SpyBufferWrapper/spy_buffer/spy_clock
/SpyBufferWrapper/spy_buffer/spy_en
/SpyBufferWrapper/spy_buffer/spy_data
/SpyBufferWrapper/spy_buffer/spy_addr
/SpyBufferWrapper/spy_buffer/spy_write_data
/SpyBufferWrapper/spy_buffer/spy_write_enable
/SpyBufferWrapper/spy_buffer/playback
/SpyBufferWrapper/spy_buffer/freeze
Internal Spy Memory Interface
/SpyBufferWrapper/spy_buffer/spy/spy_memory/clkA
/SpyBufferWrapper/spy_buffer/spy/spy_memory/addrA
/SpyBufferWrapper/spy_buffer/spy/spy_memory/weA
/SpyBufferWrapper/spy_buffer/spy/spy_memory/clkB
/SpyBufferWrapper/spy_buffer/spy/spy_memory/addrB
/SpyBufferWrapper/spy_buffer/spy/spy_memory/weB

Data read from SB Memory

SB in Freeze mode

Data written to SB Memory

Data read from SB Memory

Playback Write

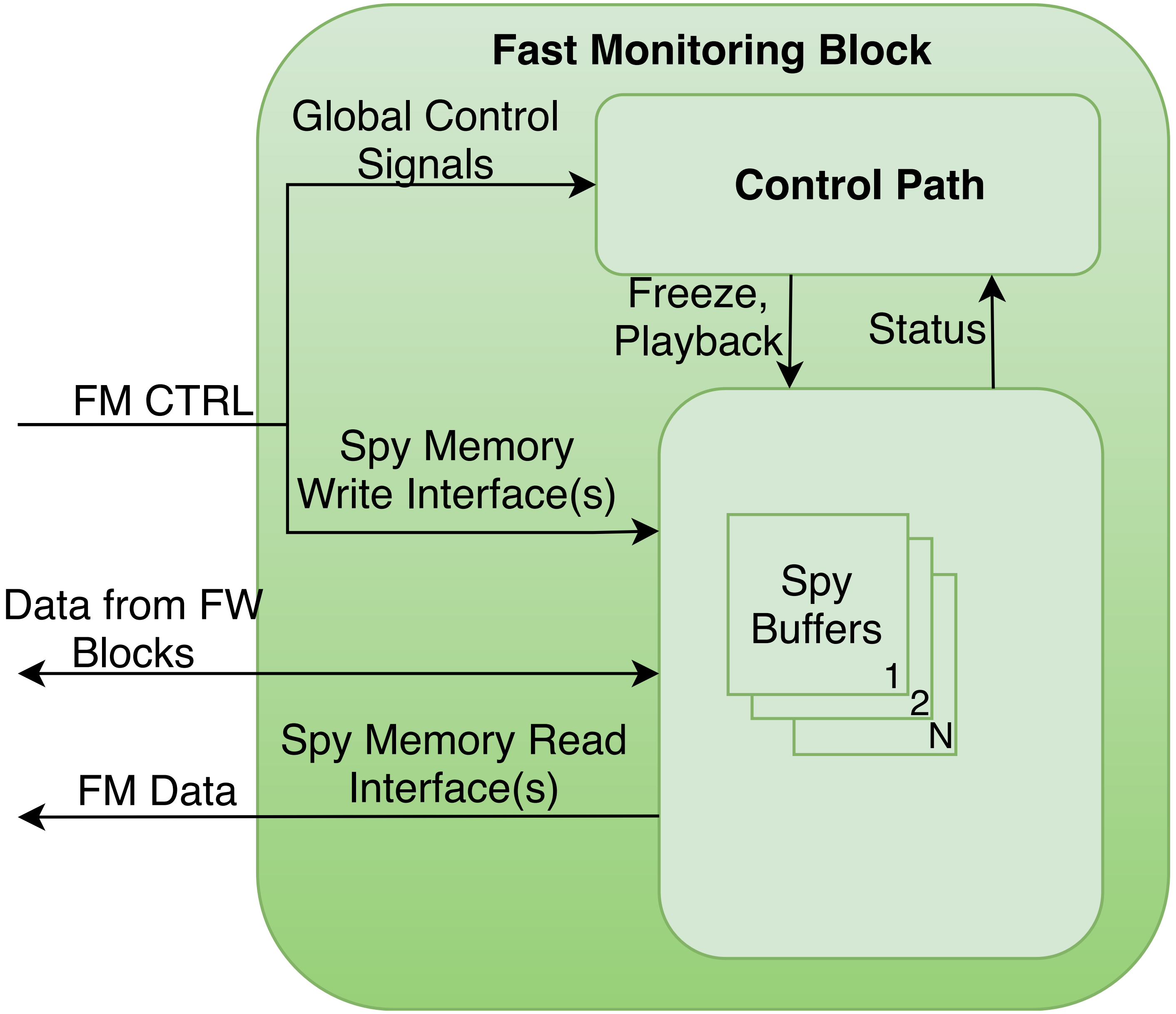
Playback Once

Playback Write

Playback Loop

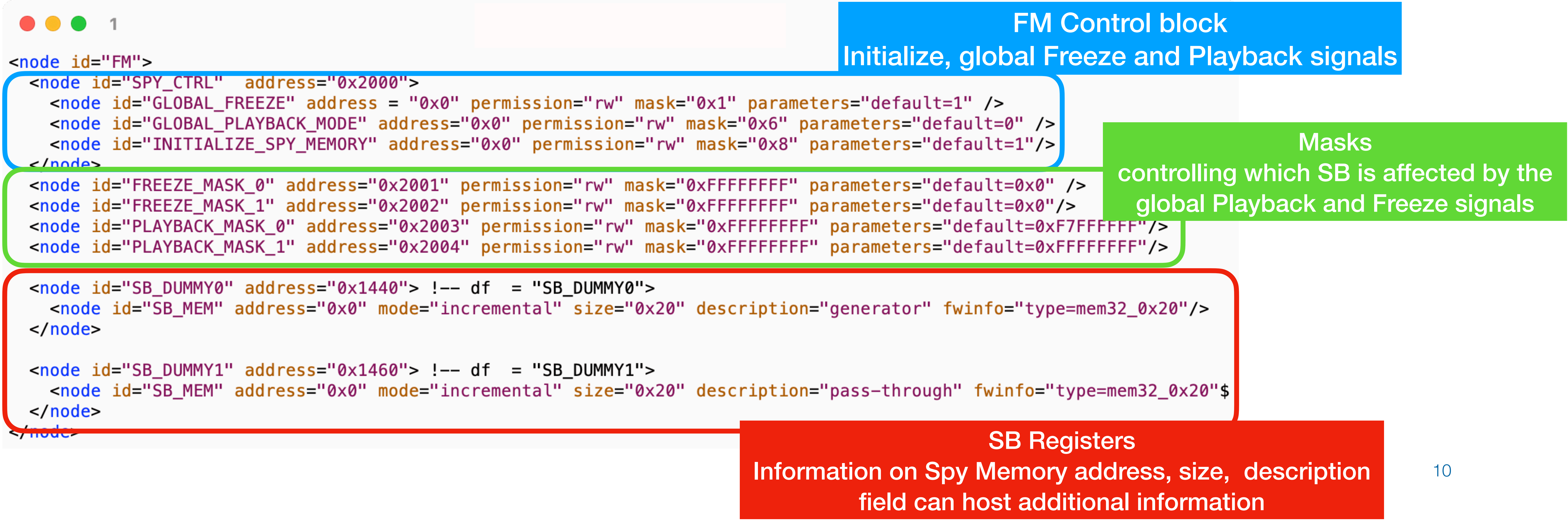
FM block implementation and resource usage

- Technology independent design: SpyBuffers and FM control are implemented using pure Verilog without the need of vendor-specific IPs
- SB memories implemented in pure RTL → Synthesis tools can optimize the implementation based on resource needs
- The FM control block defines the connections of the Control Path and needs to be implemented by the user
  - A flexible solution for many SpyBuffers defines global freeze and playback signals, as well as a masking mechanism to decide which of the SpyBuffers are affected
- Number of SpyBuffers, memory width and depth are parameters that depend on the specific use-case and affect the resource occupancy



Software module

- We tested the SpyBuffer firmware with a SoC accessing firmware registers via AXI C2C link
- A python module has been developed in order to simplify the interaction with the FM control block and SpyBuffers, based on IPbus (ipbus.web.cern.ch), (Paper: 2015 *JINST* **10** C02019)
  - IPbus is shipped with uHAL providing a C++/Python API for I/O operations on memory-mapped registers
- Registers for FM control and SpyBuffer memory access are mapped with a xml file



```
<node id="FM">
 <node id="SPY_CTRL"  address="0x2000">
   <node id="GLOBAL_FREEZE" address = "0x0" permission="rw" mask="0x1" parameters="default=1" />
   <node id="GLOBAL_PLAYBACK_MODE" address="0x0" permission="rw" mask="0x6" parameters="default=0" />
   <node id="INITIALIZE_SPY_MEMORY" address="0x0" permission="rw" mask="0x8" parameters="default=1"/>
 </node>
 <node id="FREEZE_MASK_0" address="0x2001" permission="rw" mask="0xFFFFFFFF" parameters="default=0x0" />
 <node id="FREEZE_MASK_1" address="0x2002" permission="rw" mask="0xFFFFFFFF" parameters="default=0x0"/>
 <node id="PLAYBACK_MASK_0" address="0x2003" permission="rw" mask="0xFFFFFFFF" parameters="default=0xF7FFFFFF"/>
 <node id="PLAYBACK_MASK_1" address="0x2004" permission="rw" mask="0xFFFFFFFF" parameters="default=0xFFFFFFFF"/>

 <node id="SB_DUMMY0" address="0x1440"> !-- df  = "SB_DUMMY0">
   <node id="SB_MEM" address="0x0" mode="incremental" size="0x20" description="generator" fwinfo="type=mem32_0x20"/>
 </node>

 <node id="SB_DUMMY1" address="0x1460"> !-- df  = "SB_DUMMY1">
   <node id="SB_MEM" address="0x0" mode="incremental" size="0x20" description="pass-through" fwinfo="type=mem32_0x20"$
 </node>
</node>
```

FM Control block
Initialize, global Freeze and Playback signals

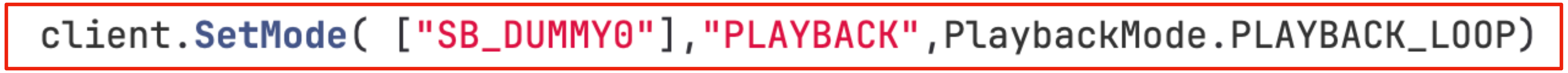
Masks
controlling which SB is affected by the global Playback and Freeze signals

SB Registers
Information on Spy Memory address, size, description field can host additional information

Software platform - FastMonitoringClient

- FastMonitoringClient allows a simple interface to FM firmware and SpyBuffers, based on the uHAL Python API

- **High-level functions is defined to execute FM operations in an easy way**
  - Bring FM in a known state (e.g. unfreeze everything and unset playback)
  - Freeze all SB or a list of SB (providing their id)
  - Report the status of each SB
  - SpyMemory I/O can be achieved via uHAL API



```
client.SetMode( ["SB_DUMMY0"],"PLAYBACK",PlaybackMode.PLAYBACK_LOOP)
```

- Masks are automatically managed by the software in a way that SpyBuffers are completely transparent to the API user

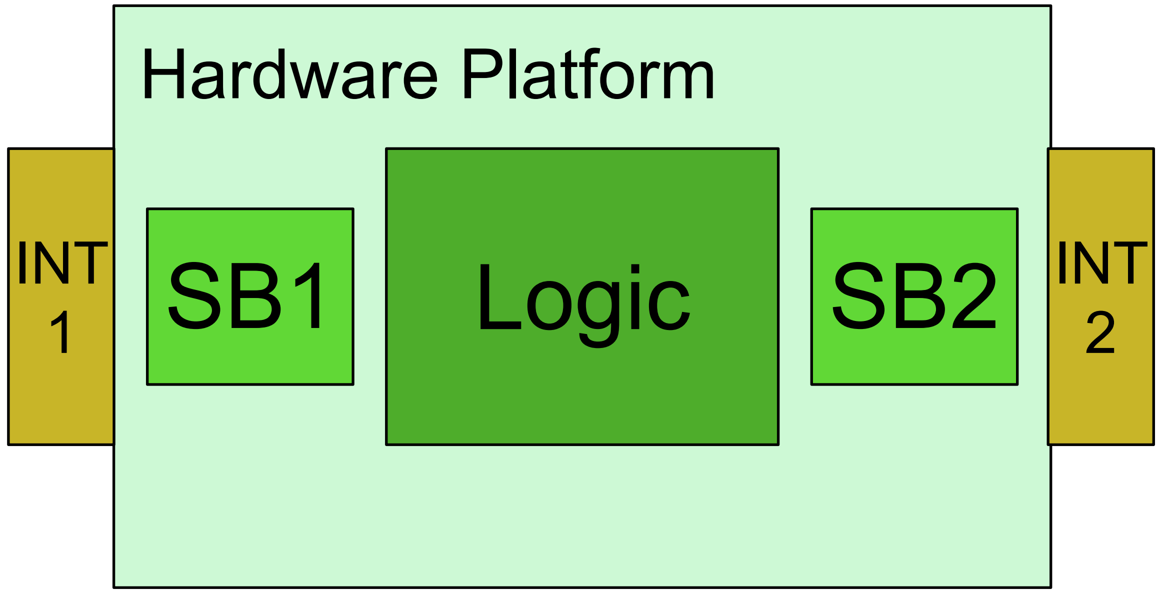
- Everything is done with the idea of minimizing the number of write/read operation. The information of SB status and masks is stored in the local memory (and updated accordingly when doing read/writes)

- SB information (id, address, etc) is extracted from the XML files

Fast Monitoring: use cases

- **Debugging of the (partial) logic block on hardware**
  - If not all the firmware blocks are implemented, SB can be used to “emulate” the missing ones

- **Perform hardware / firmware test without a fully implemented test stand**
  - Input data can be simulated offline and can be injected thanks to a SpyBuffer located after the input interface (e.g. when the detector providing input is not available)
  - Inspecting data received from the input interface before it is passed to the firmware logic

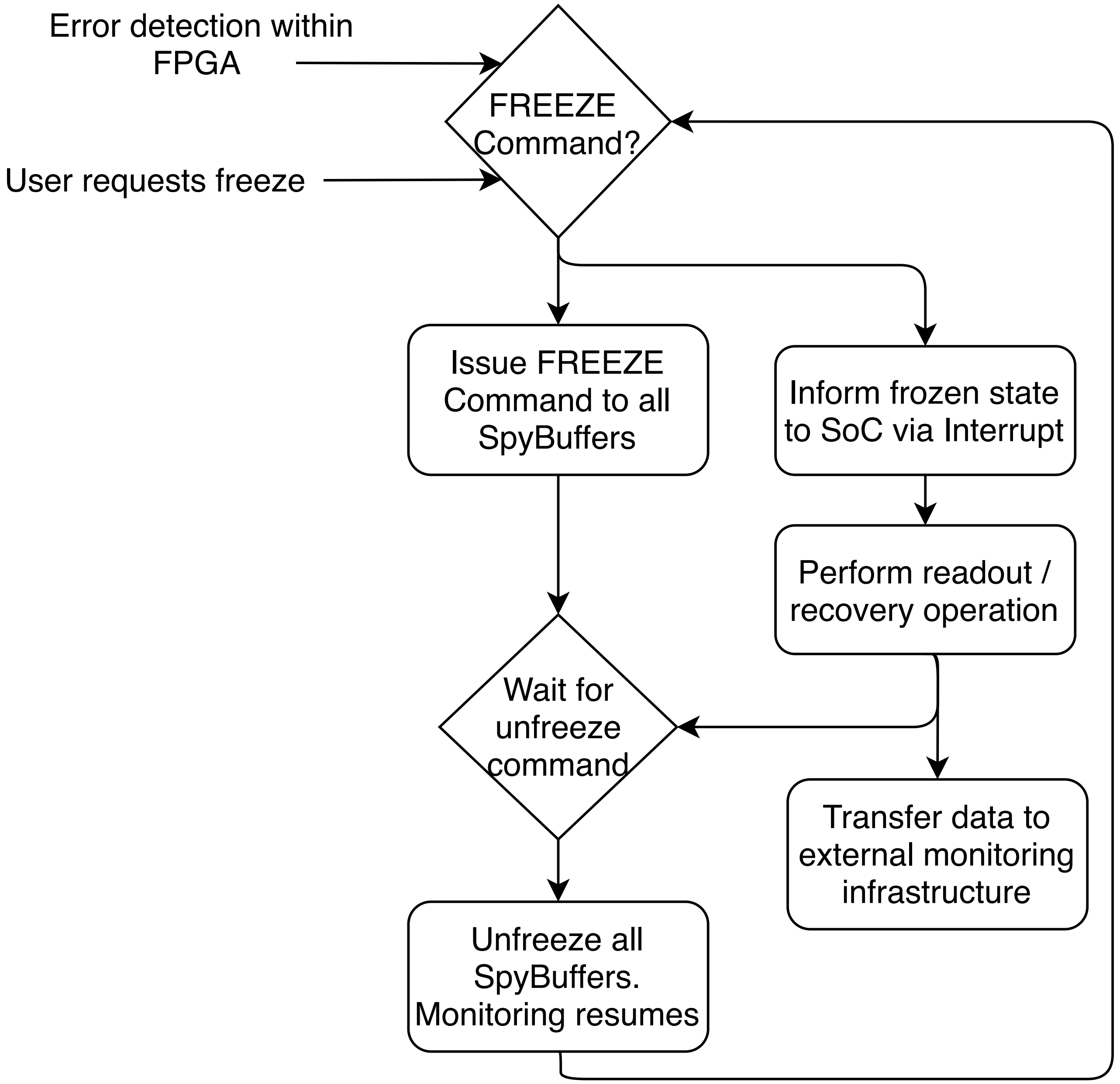


Example: Hardware platform ready to be tested, but missing input source
SB1 can be used in playback mode to emulate data from interface 1

Fast Monitoring: use cases - 2

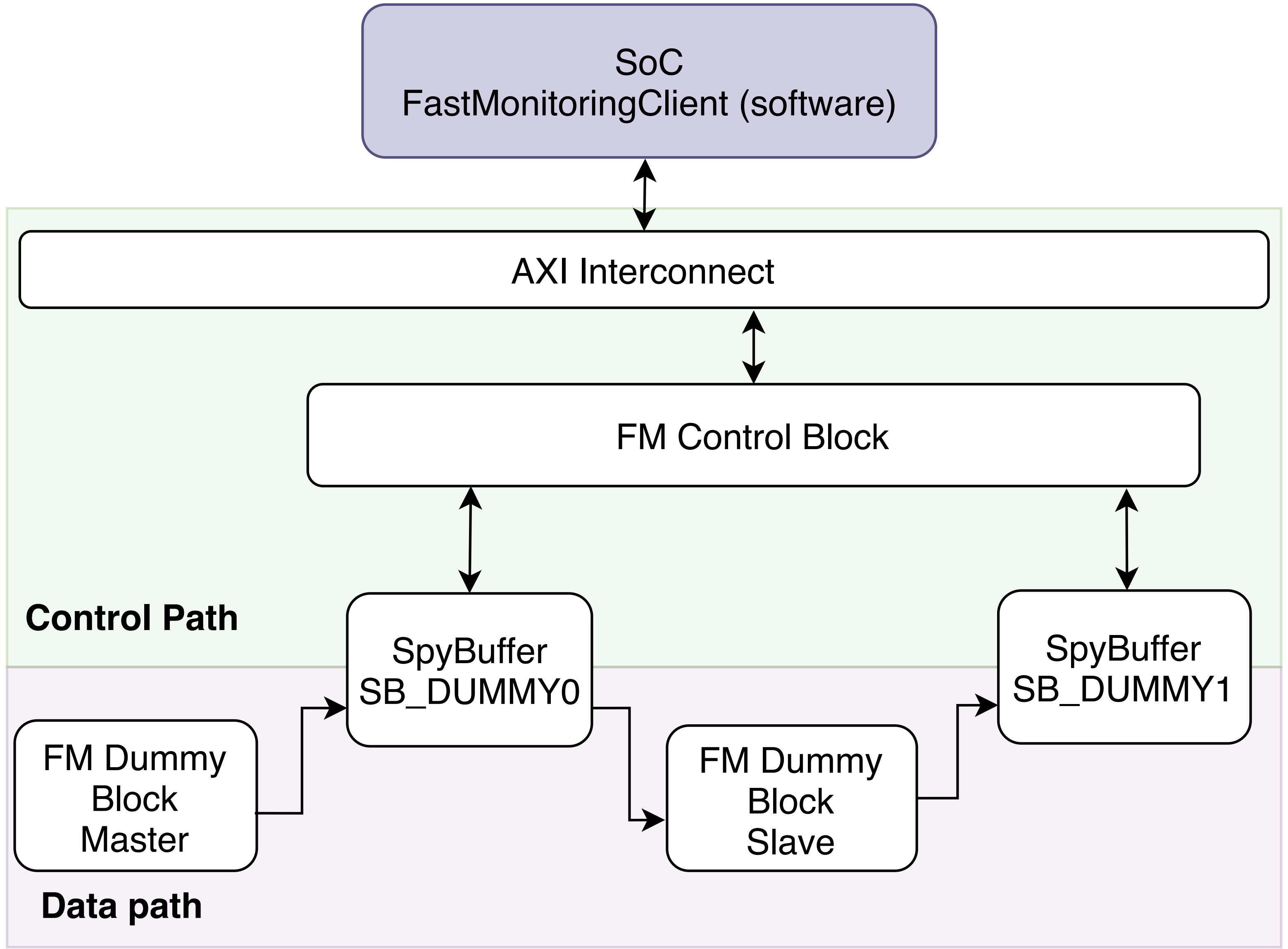
- **Powerful tool for Data Quality**
  - SpyBuffer can be placed at the output of important blocks in the firmware (e.g. where partial result are computed)
  - Extract partial information from the firmware and populate histograms to check the performance of the firmware algorithm with great detail

- **Error handling:**
  - Error states in the FPGA or in the firmware can be detected and Fast Monitoring can force the freeze status on all SpyBuffers
  - At this point some high-level analysis on the data collected from the SB can be performed
  - Data from SB can be saved for further inspection


Error detection within FPGA
User requests freeze
FREEZE Command?
Issue FREEZE Command to all SpyBuffers
Inform frozen state to SoC via Interrupt
Perform readout / recovery operation
Wait for unfreeze command
Transfer data to external monitoring infrastructure
Unfreeze all SpyBuffers. Monitoring resumes

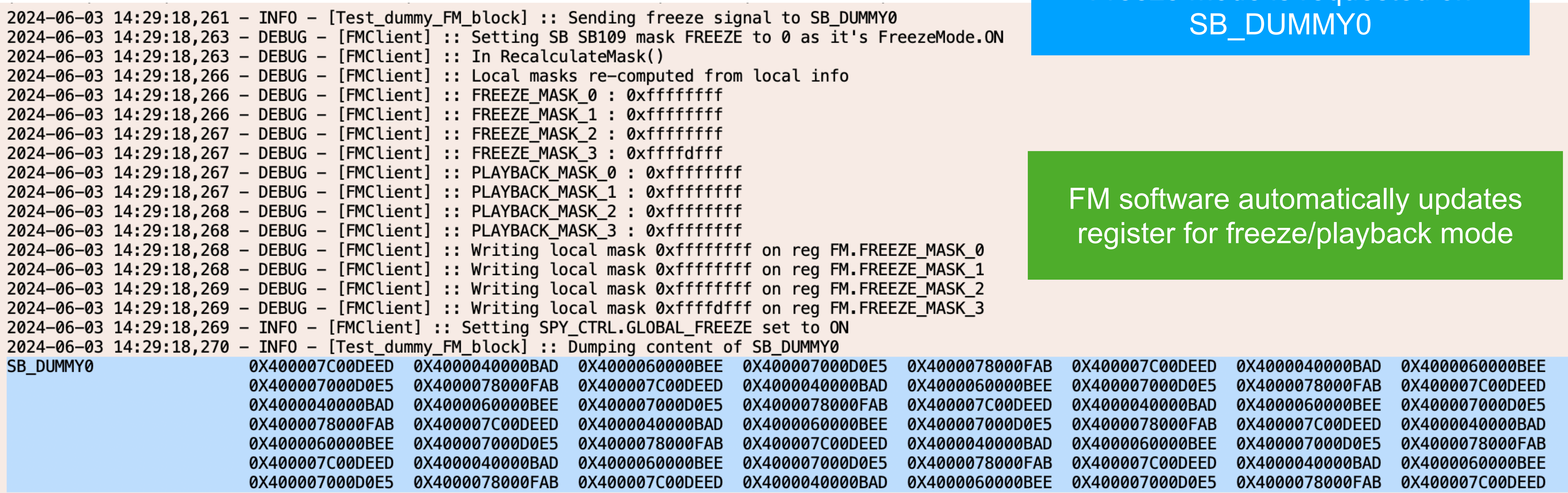
Example: Dummy FW block

- “Dummy” firmware with two blocks and two SpyBuffers
- “Master” generates a sequence of words in a loop
- “Slave” acts as a pass-through
- During normal operation, SB_DUMMY1 receives the words generated by the “Master” block



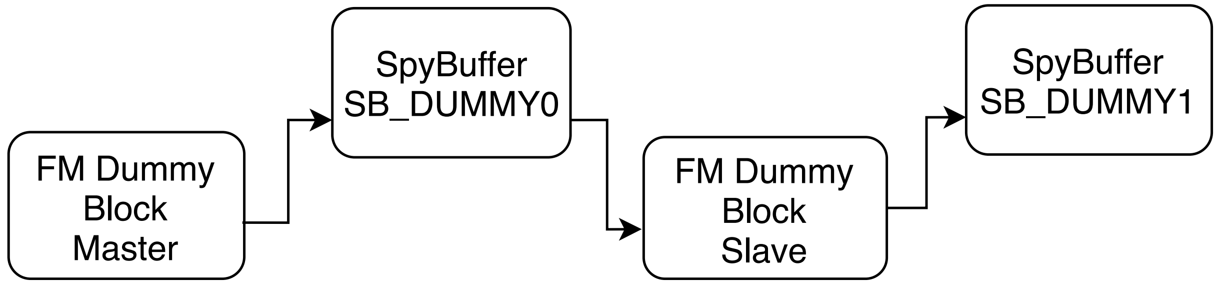
Dummy block - Freeze and readout

Freeze mode is requested on SB_DUMMY0



```
2024-06-03 14:29:18,261 - INFO - [Test_dummy_FM_block] :: Sending freeze signal to SB_DUMMY0
2024-06-03 14:29:18,263 - DEBUG - [FMClient] :: Setting SB SB109 mask FREEZE to 0 as it's FreezeMode.ON
2024-06-03 14:29:18,263 - DEBUG - [FMClient] :: In RecalculateMask()
2024-06-03 14:29:18,266 - DEBUG - [FMClient] :: Local masks re-computed from local info
2024-06-03 14:29:18,266 - DEBUG - [FMClient] :: FREEZE_MASK_0 : 0xffffffff
2024-06-03 14:29:18,266 - DEBUG - [FMClient] :: FREEZE_MASK_1 : 0xffffffff
2024-06-03 14:29:18,267 - DEBUG - [FMClient] :: FREEZE_MASK_2 : 0xffffffff
2024-06-03 14:29:18,267 - DEBUG - [FMClient] :: FREEZE_MASK_3 : 0xffffdfff
2024-06-03 14:29:18,267 - DEBUG - [FMClient] :: PLAYBACK_MASK_0 : 0xffffffff
2024-06-03 14:29:18,267 - DEBUG - [FMClient] :: PLAYBACK_MASK_1 : 0xffffffff
2024-06-03 14:29:18,268 - DEBUG - [FMClient] :: PLAYBACK_MASK_2 : 0xffffffff
2024-06-03 14:29:18,268 - DEBUG - [FMClient] :: PLAYBACK_MASK_3 : 0xffffffff
2024-06-03 14:29:18,268 - DEBUG - [FMClient] :: Writing local mask 0xffffffff on reg FM.FREEZE_MASK_0
2024-06-03 14:29:18,268 - DEBUG - [FMClient] :: Writing local mask 0xffffffff on reg FM.FREEZE_MASK_1
2024-06-03 14:29:18,269 - DEBUG - [FMClient] :: Writing local mask 0xffffffff on reg FM.FREEZE_MASK_2
2024-06-03 14:29:18,269 - DEBUG - [FMClient] :: Writing local mask 0xffffdfff on reg FM.FREEZE_MASK_3
2024-06-03 14:29:18,269 - INFO - [FMClient] :: Setting SPY_CTRL.GLOBAL_FREEZE set to ON
2024-06-03 14:29:18,270 - INFO - [Test_dummy_FM_block] :: Dumping content of SB_DUMMY0
SB_DUMMY0                0X400007C00DEED  0X4000040000BAD  0X4000060000BEE  0X400007000D0E5  0X4000078000FAB  0X400007C00DEED  0X4000040000BAD  0X4000060000BEE
                         0X400007000D0E5  0X4000078000FAB  0X400007C00DEED  0X4000040000BAD  0X4000060000BEE  0X400007000D0E5  0X4000078000FAB  0X400007C00DEED
                         0X4000040000BAD  0X4000060000BEE  0X400007000D0E5  0X4000078000FAB  0X400007C00DEED  0X4000040000BAD  0X4000060000BEE  0X400007000D0E5
                         0X4000078000FAB  0X400007C00DEED  0X4000040000BAD  0X4000060000BEE  0X400007000D0E5  0X4000078000FAB  0X400007C00DEED  0X4000040000BAD
                         0X4000060000BEE  0X400007000D0E5  0X4000078000FAB  0X400007C00DEED  0X4000040000BAD  0X4000060000BEE  0X400007000D0E5  0X4000078000FAB
                         0X400007C00DEED  0X4000040000BAD  0X4000060000BEE  0X400007000D0E5  0X4000078000FAB  0X400007C00DEED  0X4000040000BAD  0X4000060000BEE
                         0X400007000D0E5  0X4000078000FAB  0X400007C00DEED  0X4000040000BAD  0X4000060000BEE  0X400007000D0E5  0X4000078000FAB  0X400007C00DEED
```

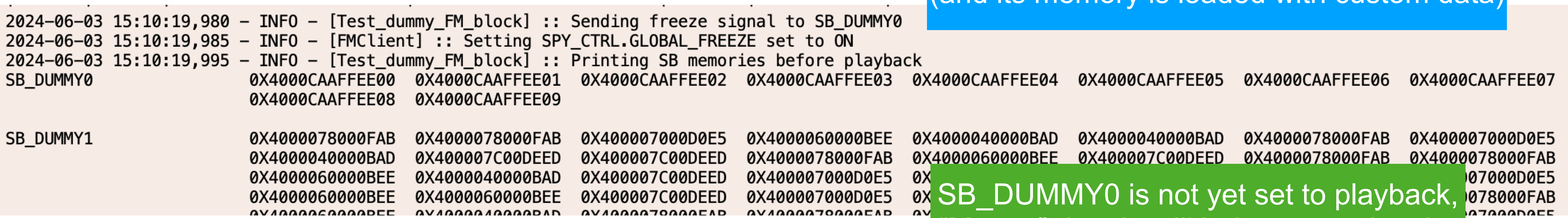
FM software automatically updates register for freeze/playback mode



Memory of SB_DUMMY0 is frozen and accessible

Dummy block - Freeze and readout 2

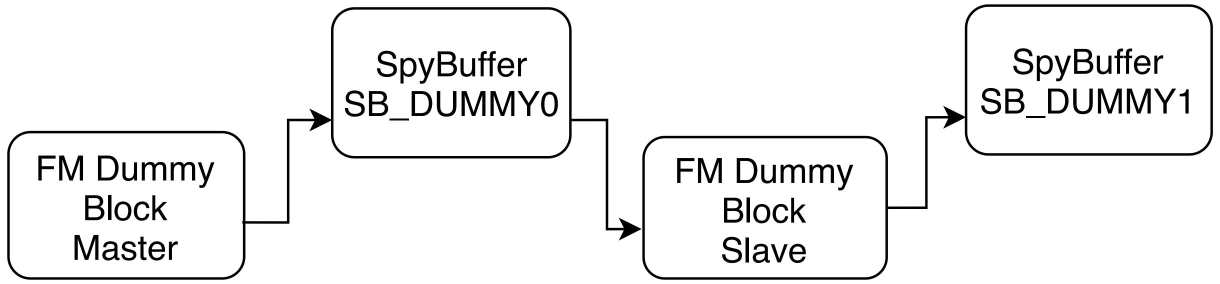
Freeze mode is requested on SB_DUMMY0 (and its memory is loaded with custom data)



```
2024-06-03 15:10:19,980 - INFO - [Test_dummy_FM_block] :: Sending freeze signal to SB_DUMMY0
2024-06-03 15:10:19,985 - INFO - [FMClient] :: Setting SPY_CTRL.GLOBAL_FREEZE set to ON
2024-06-03 15:10:19,995 - INFO - [Test_dummy_FM_block] :: Printing SB memories before playback
SB_DUMMY0                 0X4000CAAFFEE00  0X4000CAAFFEE01  0X4000CAAFFEE02  0X4000CAAFFEE03  0X4000CAAFFEE04  0X4000CAAFFEE05  0X4000CAAFFEE06  0X4000CAAFFEE07
                          0X4000CAAFFEE08  0X4000CAAFFEE09

SB_DUMMY1                 0X4000078000FAB  0X4000078000FAB  0X400007000D0E5  0X4000060000BEE  0X4000040000BAD  0X4000040000BAD  0X4000078000FAB  0X400007000D0E5
                          0X4000040000BAD  0X400007C00DEED  0X400007C00DEED  0X4000078000FAB  0X4000060000BEE  0X400007C00DEED  0X4000078000FAB  0X4000078000FAB
                          0X4000060000BEE  0X4000040000BAD  0X400007C00DEED  0X400007000D0E5  0X                                                 07000D0E5
                          0X4000060000BEE  0X4000060000BEE  0X400007C00DEED  0X400007000D0E5  0X                                                 078000FAB
                          0X4000060000BEE  0X4000040000BAD  0X4000078000FAB  0X4000078000FAB  0X                                                 07000D0E5
```

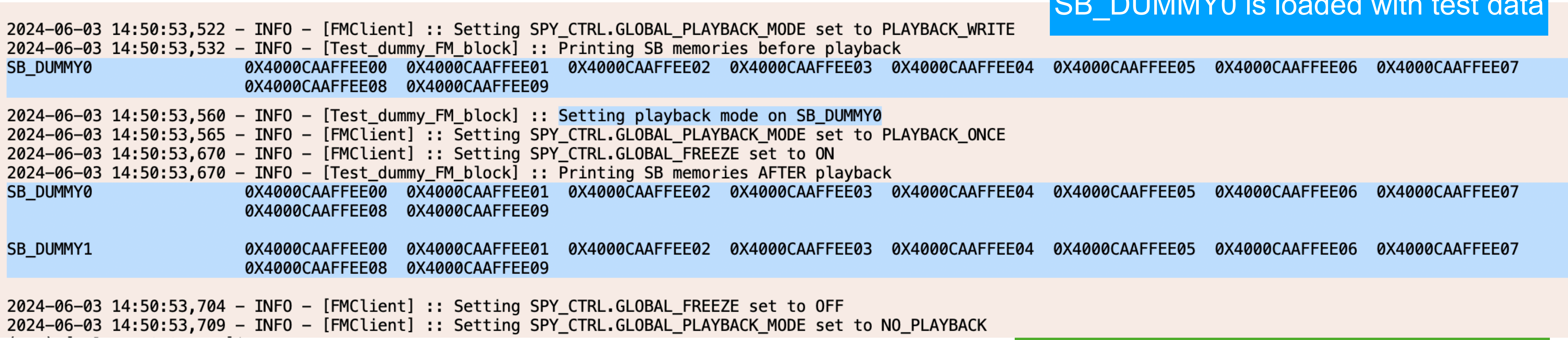
SB_DUMMY0 is not yet set to playback, “Master” data is still being passed to the “Slave” block



SB_DUMMY1 is not in “Frozen” state, data are constantly overwritten

Dummy block - Playback

Before playback mode is set, SB_DUMMY0 is loaded with test data



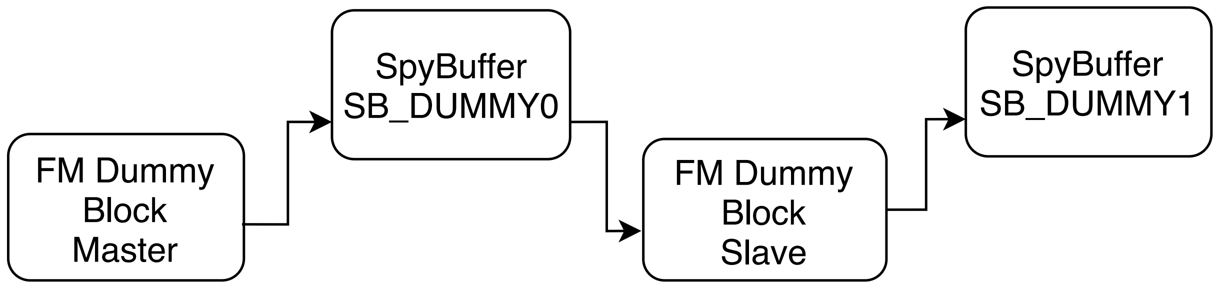
```
2024-06-03 14:50:53,522 - INFO - [FMClient] :: Setting SPY_CTRL.GLOBAL_PLAYBACK_MODE set to PLAYBACK_WRITE
2024-06-03 14:50:53,532 - INFO - [Test_dummy_FM_block] :: Printing SB memories before playback
SB_DUMMY0                0X4000CAAFFEE00  0X4000CAAFFEE01  0X4000CAAFFEE02  0X4000CAAFFEE03  0X4000CAAFFEE04  0X4000CAAFFEE05  0X4000CAAFFEE06  0X4000CAAFFEE07
                         0X4000CAAFFEE08  0X4000CAAFFEE09

2024-06-03 14:50:53,560 - INFO - [Test_dummy_FM_block] :: Setting playback mode on SB_DUMMY0
2024-06-03 14:50:53,565 - INFO - [FMClient] :: Setting SPY_CTRL.GLOBAL_PLAYBACK_MODE set to PLAYBACK_ONCE
2024-06-03 14:50:53,670 - INFO - [FMClient] :: Setting SPY_CTRL.GLOBAL_FREEZE set to ON
2024-06-03 14:50:53,670 - INFO - [Test_dummy_FM_block] :: Printing SB memories AFTER playback
SB_DUMMY0                0X4000CAAFFEE00  0X4000CAAFFEE01  0X4000CAAFFEE02  0X4000CAAFFEE03  0X4000CAAFFEE04  0X4000CAAFFEE05  0X4000CAAFFEE06  0X4000CAAFFEE07
                         0X4000CAAFFEE08  0X4000CAAFFEE09

SB_DUMMY1                0X4000CAAFFEE00  0X4000CAAFFEE01  0X4000CAAFFEE02  0X4000CAAFFEE03  0X4000CAAFFEE04  0X4000CAAFFEE05  0X4000CAAFFEE06  0X4000CAAFFEE07
                         0X4000CAAFFEE08  0X4000CAAFFEE09

2024-06-03 14:50:53,704 - INFO - [FMClient] :: Setting SPY_CTRL.GLOBAL_FREEZE set to OFF
2024-06-03 14:50:53,709 - INFO - [FMClient] :: Setting SPY_CTRL.GLOBAL_PLAYBACK_MODE set to NO_PLAYBACK
```

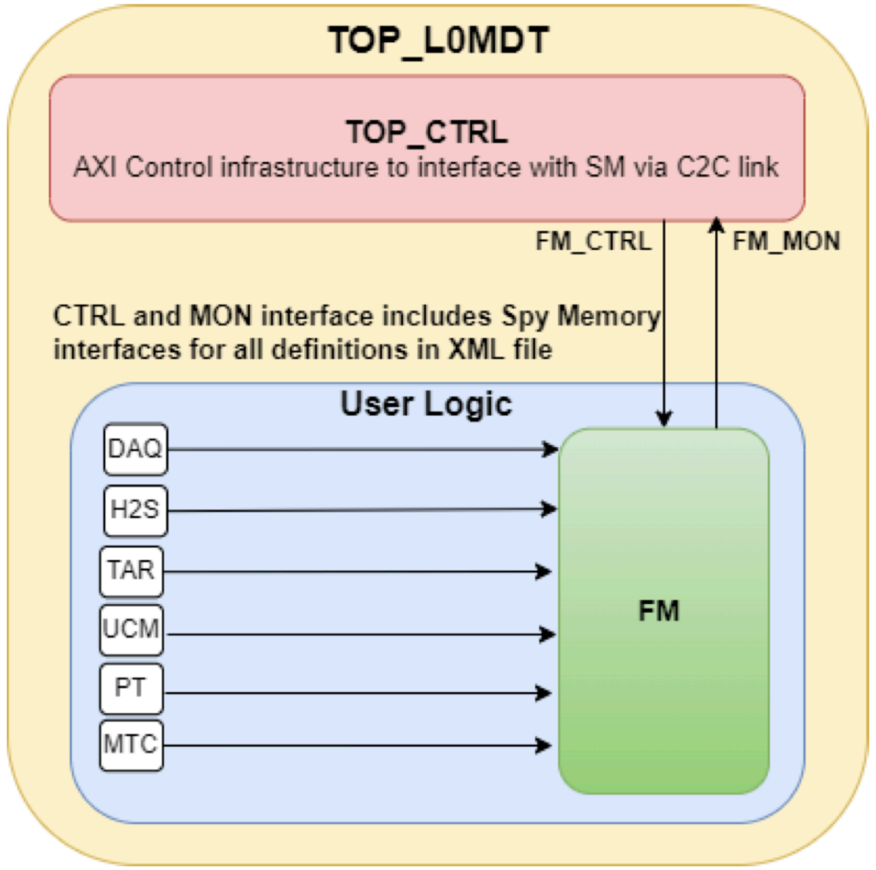
After playback the same content is available in SB_DUMMY1

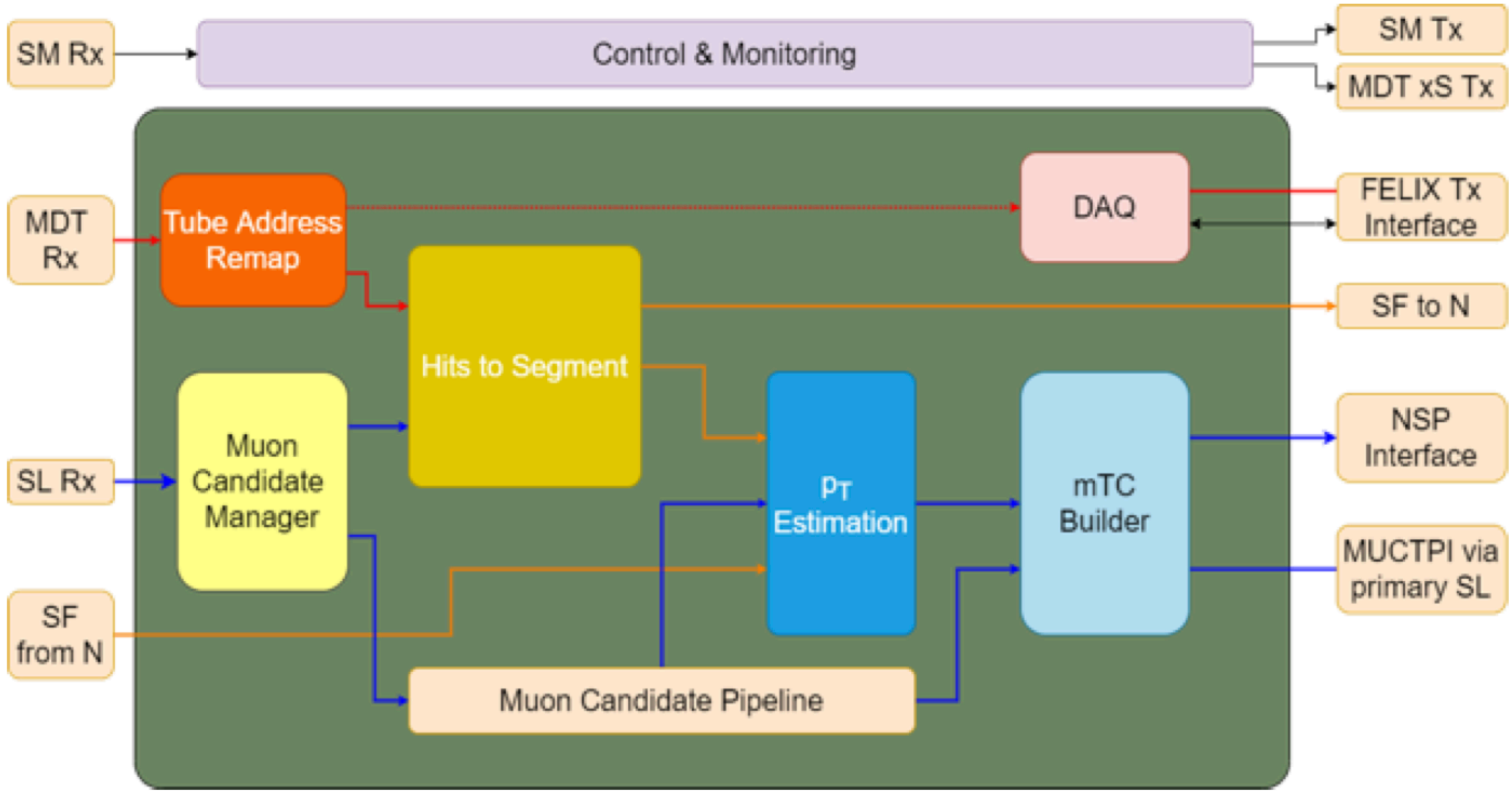


PLAYBACK_LOOP mode has also been tested

Applications: ATLAS MDT-TP

- ATLAS implements Fast Monitoring in the MDT Trigger Processor (for Phase-2 upgrade of the Muon Trigger system)
- During the development stage FM is used for firmware validation and testing
- During HL-LHC operation this will be used to produce Data Quality histograms and as a expert tool for debugging


TOP_L0MDT
TOP_CTRL
AXI Control infrastructure to interface with SM via C2C link
FM_CTRL
FM_MON
CTRL and MON interface includes Spy Memory interfaces for all definitions in XML file
User Logic
DAQ
H2S
TAR
UCM
PT
MTC
FM



Summary

- Fast Monitoring and SpyBuffers enable a flexible way to perform debugging of complicated firmware algorithms

- **Verilog implementation and software resources** **https://gitlab.cern.ch/spybuffer/spybuffer**
  - This git repository contains also a series of unit tests of the basic data flow, spy memory readout, playback mode including an optional FIFO
  - Tests are written in Python using cocotb and can be run with a supported simulator (Questa, Cadence)

Backup

Example resource usage

- Resource usage on Xilinx VU13P for a single SpyBuffer
  - Memory width 128 and depth 512 bits
  - VU13P BRAM size is 36 kbit
  - Max data width supported 32 bit
  - 128/32 = 4 BRam

CLB		98
	CLB Registers	126
	CLB LUTs	98
Block RAM		4

SpyBuffer Parameter	Setting
DATA_WIDTH_A	128
DATA_WIDTH_B	32
SPY_MEM_WIDTH_A	9
SPY_MEM_WIDTH_B	11
PASSTHROUGH	0