NeoSpectra™ SWS62231 – Development Kits' Developer Manual

Information in this document is provided in connection with Si-Ware Systems products. These materials are provided by Si-Ware Systems as a service to its customers and may be used for informational purposes only. Si-Ware Systems assumes no responsibility for errors or omissions in these materials. Si-Ware Systems may make changes to its products, specifications, and product descriptions at any time, without notice. Si-Ware Systems makes no commitment to update the information and shall have no responsibility whatsoever for conflicts, incompatibilities, or other difficulties arising from future changes to its products and product descriptions. No license, express or implied, by estoppels or otherwise, to any intellectual property rights is granted by this document. Except as may be provided in Si-Ware Systems' Terms and Conditions of Sale for such products, Si-Ware Systems assumes no liability whatsoever.

Copyright

Contacts

For technical assistance, please contact:

Si-Ware Systems
3, Khaled Ibn Al-Waleed St.
Sheraton, Heliopolis
Cairo 11361, Egypt

Tel.: + 20 222 68 47 04
Email: neospectra.support@si-ware.com

Trademarks

NeoSpectra™ and SpectroMOST™ are trademarks of Si-Ware Systems

Contents

Developer modes

Chapter 1 Mode 1: Using SPI Interface

NeoSpectra Micro DVK allows direct communication through SPI with SPI master. SPI driver can be built on the Raspberry PI board or on any other development boards which contain SPI master interface.
The description of the SPI slave interface is explained in this section.

1. Requirements

- NeoSpectra Micro DVK.
- Customer board connected to the NeoSpectra Micro DVK through SPI.

2. Interface

Remove the raspberry pi board entirely and use NeoSpectra Micro DVK board only by connecting it to an external power supply and interface with it through SPI* interface and IOs as shown in Figure.1: NeoSpectra Micro DVK Pins .

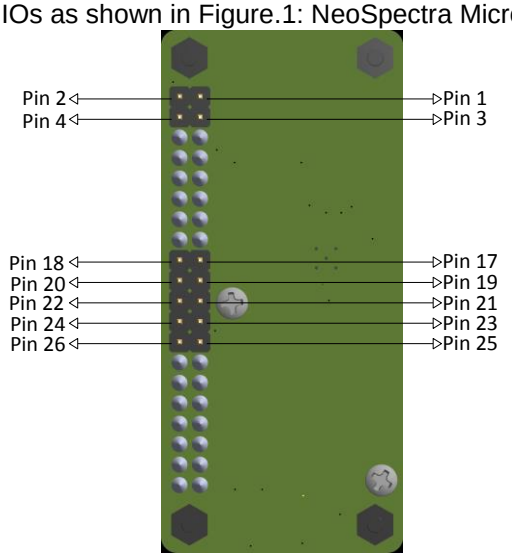


Figure.1: NeoSpectra Micro DVK Pins

The pins assignment of NeoSpectra Micro DVK is as shown in the following table.

Pin number	Function
Pin 1	Not connected
Pin 2	VDD +5v
Pin 3	GND
Pin 4	VDD +5v
Pin 17	Not connected
Pin 18	Data Ready Pin
Pin 19	SPI MOSI
Pin 20	GND
Pin 21	SPI MISO
Pin 22	Interrupt Pin
Pin 23	SPI Clk
Pin 24	SPI CSB
Pin 25	GND
Pin 26	External Trigger Pin

* The maximum SPI rate to be used is 1MHz

3. Registers Description

(= Register Possible Values)

Register Name	Width (Bits)	Description	Type	Address	Offset (Bits)	Fixed-Point Quantization Length (Bits)†
MODULE_ID	64	DVK ID.	R	0	0	
AUTO_INCB	1	Address Auto increment enable (Active Low) 0 → Auto increment feature is enabled, which means multiple addresses of register file can be accessed in a single frame. 1 (default) → Auto increment feature is disabled, which means only one address can be accessed in a single frame.	R/W	12	0	
SNGL_CNT_MODE	4	Choose between the single and continuous modes of scanning. The continuous mode boosts the speed of scanning during specific period. (only valid during ACQUIRE_PSD & RUN_SPECTRUM _SAMPLE commands) 0 (default) → single mode scanning. 4 → continuous mode scanning.	R/W	13	1	
XZP	2	Selects the zero padding option which is used to specify FFT number of points: 0 (default) or 1 → 1X (8k points). 2 → 2X (16k points). 3 → 4X (32k points).	R/W	13	5	
EN_COMMON_WAVE	1	Enables the Common wave number feature (linear interpolation) 0 (default) → disabled. 1 → enabled.	R/W	13	7	
UNIT_CONV	1	Selects the unit of the Wavenumber 0 (default)→ wavenumber. 1→ wavelength.	R/W	14	0	
OPT_GAIN_SET_SEL	2	Selects which optical gain settings to use during the scan 0 (default) → Flashed optical gain settings 1 → last calculated optical gain settings 2 → External optical gain settings	R/W	14	1	
WIN_SEL	3	Selects the used window for Apodization 0 (default) → No window is applied (boxcar). 1 → Gaussian. 2 → Happ-Genzel. 3 → Lorenz. 4 → External coefficients.	R/W	14	3	
SCAN_TIME	24	This register defines the scan time in milli-seconds.	R/W	16	0	
PSD_NO_POINTS	13	Selects the number of points of the PSD and corresponding wave number (Up to 4K samples) *valid only in case EN_COMMON_WAVE = 1.	R/W	20	0	
PSD_LENGTH	13	Output PSD length (in samples) (Up to 4 K samples).	R	22	0	
INITIATE_OPERATION	8	Writing this register initiates a certain operation that corresponds to the written	R/W	24	0	

† This field indicates whether the register represent a fixed-point value or a normal value. If it's a fixed-point value , then
the corresponding double-precision number = register value / $2^{quatization\_length}$

		code.				
1 → ACQUIRE_PSD		Initiates PSD acquisition operation.				
2 → RUN_SELF_CORR		Initiates the self-correction routine but will not write the output on Flash. The results will be stored on a volatile SRAM. It will not be kept after power down unless written on flash by user.				
3 → RUN_REF_MTR_CORR_BG		Initiates background reading taken to perform reference material correction.				
4 → RUN_REF_MTR_CORR		Initiates reference material sample reading and run reference material correction routine. *must be done after RUN_REF_MTR_CORR_BG command.				
5 → RUN_OPT_GAIN_ADJST		Initiates the Optical gain adjustment routine. After its completion, the results will be stored on OPT_GAIN_SET_OUT register.				
7 → WR_WIN_REQ		External apodization window write request.				
8 → RD_PSD_WVN_REQ		PSD or WVN read request.				
10 → RD_WIN_SMPL_REQ		Generated apodization window samples read request.				
11 → PGM_SELF_CORR_COEFF		Stores the results of Self Correction on Flash.				
12 → PGM_REF_MTR_COEFF		Stores the results of Reference Material Correction on Flash.				
13 → PGM_OPT_GAIN_SET		Stores the results of Optical gain adjustment on Flash.				
15 → RESTORE_FACTORY_CORR		Restore the original values of Self Correction, Reference Material Correction and Optical Gain Adjustment. And clear any values the user previously stored on Flash.				
16 → RUN_REFLECTANCE _BG		Performs a background scan.				
17 → RUN_REFLECTANCE_SAMPLE		Performs a sample scan and calculate the absorbance. *must be done after RUN_REFLECTANCE _BG command.				
ABORT_OPERATION	1	This register is used to exit the continuous mode scanning and will put the DVK to an idle state. It can be used to cancel any ongoing operation.	WO	28	0	
SPCTRM_DATA_OUT	8	Calculated Spectrum.	R	32	0	33
DVK_VERSION	32	Version of the software on the DVK.	R	36	0	
WAVE_NUM_DATA_OUT	8	Calculated Wavenumber or Wavelength.	R	40	0	30
SOURCE_LAMPS_COUNT	8	Number of lamps to operate from the source. 1 → one active lamp. 2 (default) → two active lamps.	R/W	41	0	
SOURCE_LAMP_SEL	8	In case the value written in SOURCE_LAMPS_COUNT is 1, this register determines which lamp to operate 0 → lamp0 is active. 1 → lamp1 is active.	R/W	42	0	
SOURCE_T1	8	Delay time after opening the source in 50 ms unit. 0 (default) → 0 ms. 1 → 50 ms. 2 → 100ms. etc…	R/W	44	0	
SOURCE_T2	8	Delay time before closing the source in 50 ms unit. 0 (default) → 0 ms. 1 → 50 ms. 2 → 100ms. etc…	R/W	45	0	

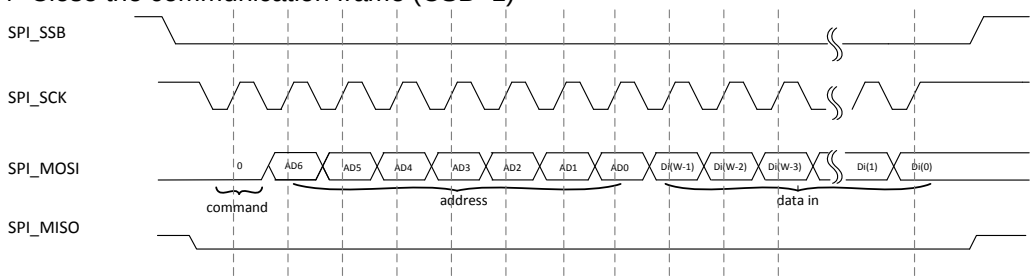
SOURCE_DELTA_T	8	Delay time between opening/closing the two lamps of the source in 50 ms unit. 0 → 0 ms. 1 → 50 ms. 2 (default) → 100ms. etc…	R/W	46	0	
SOURCE_AUTO_MODE	1	Writing 0 to this register will make the source switched off even during scans. 0 → Reserved. 1 (default) → Automatic mode of the source.	R/W	47	0	
GENERIC_DATA_OUT_LEN	16	Other data out Length register (in samples).	R	48	0	
GENERIC_DATA_OUT	8	Other generic data output register.	R	50	0	Based on data type
STATUS	32	Status of the requested operation. To be checked after OPERATION_RDY = 1 to know whether the requested operation completed successfully or not. 0 (default)→ No error. Any other value → error.	R	56	0	
OPERATION_RDY	1	When '1' Indicates no operation is in progress. User shouldn't perform any action while register is 0. 0 → DVK is busy. User should wait. 1 (default) → DVK is ready for actions.	R	60	0	
INTRPT	1	Indicates whether an error has occurred in the requested operation or not. 0 (default) → No error. 1 → error.	R	60	1	
REF_MTR_WELL_0	32	Reference material well 0 used in reference material correction. *(if = zero → will not be used)	R/W	64	0	20
REF_MTR_WELL_1	32	Reference material well 1 used in reference material correction. *(if = zero → will not be used)	R/W	68	0	20
REF_MTR_WELL_2	32	Reference material well 2 used in reference material correction. *(if = zero → will not be used)	R/W	72	0	20
REF_MTR_WELL_3	32	Reference material well 3 used in reference material correction. *(if = zero → will not be used)	R/W	76	0	20
REF_MTR_WELL_4	32	Reference material well 4 used in reference material correction. *(if = zero → will not be used)	R/W	80	0	20
GENERIC_DATA_IN_LEN	16	Determines the length of external stream in vector (in samples)	R/W	84	0	
GENERIC_DATA_IN	8	Incrementally accepts different stream in data after requesting its corresponding command.	R/W	86	0	Based on data type
OPT_GAIN_SET_EXT	16	This register defines the settings of the optical gain adjustment routine and is divided as follows: Bits 0-2: Current range settings. Bits 3-5: PGA1 settings. Bits 6-8: PGA2 settings. Bits 9-15: reserved.	R/W	92	0	
OPT_GAIN_SET_OUT	16	This register holds the output of the Optical gain adjustment routine.	R	94	0	

4. SPI Slave User Guide

SPI_MODE (SPI is selected when CSB=0)

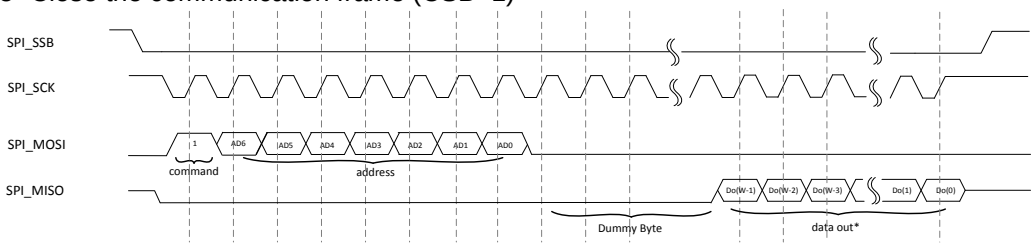
For user to write a certain register:

1- Opens a communication frame (CSB=0)
2- In the first 8 clock cycles, transmit the address of the register (7 bits) starting with MSB concatenated with '0' in the first bit
3- Transmit the required data to be written in the following bytes
4- Close the communication frame (CSB=1)



For user to read a certain register:

1- Opens a communication frame (CSB=0)
2- Transmit the address of the register (7 bits) concatenated with '1' in the first bit in the first 8 clock cycles
3- Transmit a number of dummy bytes equal to the number of bytes to be read in the following frame bytes + 1.
4- The data to be read will be available starting from the second byte of the dummy bytes (i.e. the third byte in total)
5- Close the communication frame (CSB=1)



General Rules

- Writing in register file is valid only as long as OPERATION_RDY = 1 except for ABORT_OPERATION
- Once a new operation is requested (INITIATE_OPEARTION register is written) OPERATION_RDY goes to '0' and writing is not valid until operation is ended.
- Streaming in/out data has to be done in one single frame with AUTO_INCB = 1
- If ACQUIRE_PSD operation is initiated in one of the continuous modes, it will keep running infinitely and all other operations can't be requested again unless operation is aborted through ABORT_OPERATION register. This takes NeoSpectra Micro DVK again to the idle state.
- INTRPT register/pin can be used to track status for the on-going operation or after operation ends.

For user to request ANY operation: (General Rule)

1- Poll on OPERATION_RDY register until it becomes '1'
2- Write the needed configuration registers (resolution, scan time…)
3- Write the required command in INITIATE_OPERATION register.
4- Poll on OPERATION_RDY register to be '1' or wait for OPERATION_RDY interrupt indicating end of operation

5. Detailed Examples of Operations

Below are examples of different user operations described in detailed steps.

- **Acquire_PSD operation and read the results directly:**

1- Poll on OPERATION_RDY register until it becomes '1' or wait for the OPERATION_RDY pin
2- Write the needed configuration registers (scan time,...)
3- Write AUTO_INCB = 1
4- Write "ACQUIRE_PSD" code in INITIATE_OPERATION register
5- Poll on OPERATION_RDY register till it becomes '1' or wait for the OPERATION_RDY pin (If INTRPT reg/pin is set during this waiting period, this indicates a warning, read STATUS register to check it)
6- Read STATUS register to check the status of the operation
7- Read PSD_LENGTH register
8- Read the PSD from SPCTRM_DATA_OUT register successively in one frame with the number of samples determined by PSD_LENGTH
9- Close the communication frame
10- Read the Wave number vector from WAVE_NUM_DATA_OUT register successively in one frame with the number of samples determined by PSD_LENGTH
11- Close the communication frame

- **Acquire_PSD operation and read the results directly with continuous mode:**

1- Poll on OPERATION_RDY register until it becomes '1' or wait for the OPERATION_RDY pin
2- Write the needed configuration registers (scan time,...)
3- Write '4' to SNGL_CNT_MODE register for selecting continuous scanning mode
4- Write AUTO_INCB = 1
5- Write "ACQUIRE_PSD" code in INITIATE_OPERATION register
6- Poll on OPERATION_RDY register till it becomes '1' or wait for the OPERATION_RDY pin (If INTRPT reg/pin is set during this waiting period, this indicates a warning, read STATUS register to check it)
7- Read STATUS register to check the status of the operation
8- Read PSD_LENGTH register
9- Read the PSD from SPCTRM_DATA_OUT register successively in one frame with the number of samples determined by PSD_LENGTH
10- Close the communication frame
11- Read the Wave number vector from WAVE_NUM_DATA_OUT register successively in one frame with the number of samples determined by PSD_LENGTH
12- Close the communication frame
13- After reading both PSD & WVN, OPERATION_RDY register (and pin) should automatically go to '0'. So the user should loop on the steps from 6→13 to acquire further scans without the need for initiating new commands
14- For **exiting** the continuous mode, write SNGL_CNT_MODE = '0'. Then acquire one last PSD & WVN. OPERATION_RDY register (and pin) should stay '1' after that last acquisition until any further command is requested

- **RUN_OPT_GAIN_ADJST operation to apply gain adjustment routine:**

1- Poll on OPERATION_RDY register until it becomes '1' or wait for the OPERATION_RDY pin
2- Write the needed configuration registers (scan time,...)
3- Write AUTO_INCB = 1
4- Write "RUN_OPT_GAIN_ADJST" code in INITIATE_OPERATION register
5- Poll on OPERATION_RDY register till it becomes '1' or wait for the OPERATION_RDY pin (If INTRPT reg/pin is set during this waiting period, this indicates a warning, read STATUS register to check it)
6- Read STATUS register to check the status of the operation
7- Write OPT_GAIN_SET_SEL = 1 to use the calculated gain value in the upcoming measurements.

- **PGM_OPT_GAIN_SET operation to store the result of gain adjustment routine on flash:**

1- Poll on OPERATION_RDY register until it becomes '1' or wait for the OPERATION_RDY pin.
3- Write AUTO_INCB = 1
4- Write "PGM_OPT_GAIN_SET" code in INITIATE_OPERATION register
5- Poll on OPERATION_RDY register till it becomes '1' or wait for the OPERATION_RDY pin (If INTRPT reg/pin is set during this waiting period, this indicates a warning, read STATUS register to check it)
6- Read STATUS register to check the status of the operation
7- Write OPT_GAIN_SET_SEL = 0 to use the flashed gain value in the upcoming measurements.

- **RUN_SELF_CORR operation to apply self-correction routine:**

1- Poll on OPERATION_RDY register until it becomes '1' or wait for the OPERATION_RDY pin
2- Write the needed configuration registers (scan time,...)
3- Write AUTO_INCB = 1
4- Write "RUN_SELF_CORR" code in INITIATE_OPERATION register
5- Poll on OPERATION_RDY register till it becomes '1' or wait for the OPERATION_RDY pin (If INTRPT reg/pin is set during this waiting period, this indicates a warning, read STATUS register to check it)
6- Read STATUS register to check the status of the operation

- **PGM_SELF_CORR_COEFF operation to store the result of self-correction routine on flash:**

1- Poll on OPERATION_RDY register until it becomes '1' or wait for the OPERATION_RDY pin
2- Write the needed configuration registers (scan time,...)
3- Write AUTO_INCB = 1
4- Write "PGM_SELF_CORR_COEFF" code in INITIATE_OPERATION register
5- Poll on OPERATION_RDY register till it becomes '1' or wait for the OPERATION_RDY pin (If INTRPT reg/pin is set during this waiting period, this indicates a warning, read STATUS register to check it)
6- Read STATUS register to check the status of the operation

- **RUN_REF_MTR_CORR_BG operation to apply reference material correction routine (background scan):**

1- Poll on OPERATION_RDY register until it becomes '1' or wait for the OPERATION_RDY pin.
2- Write the needed configuration registers (scan time,...)
3- Write "RUN_REF_MTR_CORR_BG" code (3) in INITIATE_OPERATION register
4- Poll on OPERATION_RDY register till it becomes '1' or wait for the OPERATION_RDY pin (If INTRPT reg/pin is set during this waiting period, this

indicates a warning, read STATUS register to check it)
5- Read STATUS register to check the status of the operation

- **RUN_REF_MTR_CORR operation to apply reference material correction routine (reference material scan):**

1- Poll on OPERATION_RDY register until it becomes '1' or wait for the OPERATION_RDY pin.
2- Write the needed configuration registers (scan time,...)
3- Write peaks' wavelengths of the used reference material (up to 5 wavelengths).
   - REF_MTR_WELL_0 = reference material peak0 wavelength.
   - REF_MTR_WELL_1 = reference material peak1 wavelength (if any, 0 otherwise)
   - REF_MTR_WELL_2 = reference material peak2 wavelength (if any, 0 otherwise)
   - REF_MTR_WELL_3 = reference material peak3 wavelength (if any, 0 otherwise)
   - REF_MTR_WELL_4 = reference material peak4 wavelength (if any, 0 otherwise)
4- Write "RUN_REF_MTR_CORR" code (4) in INITIATE_OPERATION register
5- Poll on OPERATION_RDY register till it becomes '1' or wait for the OPERATION_RDY pin (If INTRPT reg/pin is set during this waiting period, this indicates a warning, read STATUS register to check it)
6- Read STATUS register to check the status of the operation

- **PGM_REF_MTR_COEFF operation to store the result of reference material correction routine on flash:**

1- Poll on OPERATION_RDY register until it becomes '1' or wait for the OPERATION_RDY pin
2- Write "PGM_REF_MTR_COEFF" code (12) in INITIATE_OPERATION register
3- Poll on OPERATION_RDY register till it becomes '1' or wait for the OPERATION_RDY pin (If INTRPT reg/pin is set during this waiting period, this indicates a warning, read STATUS register to check it)
4- Read STATUS register to check the status of the operation

- **RESTORE_FACTORY_CORR operation to restore correction and optical gain settings to the factory settings:**

1- Poll on OPERATION_RDY register until it becomes '1' or wait for the OPERATION_RDY pin
2- Write "RESTORE_FACTORY_CORR" code (15) in INITIATE_OPERATION register
3- Poll on OPERATION_RDY register till it becomes '1' or wait for the OPERATION_RDY pin (If INTRPT reg/pin is set during this waiting period, this indicates a warning, read STATUS register to check it)
4- Read STATUS register to check the status of the operation

- **Read PSD or WVN with a read request:**

In order to get back to read PSD or WVN after another operation is requested, user doesn't have to repeat ACQUIRE_PSD operation given that it has done before. This is done through RD_PSD_WVN_REQ command
1- Poll on OPERATION_RDY register until it becomes '1' or wait for the OPERATION_RDY interrupt pin
2- Write AUTO_INCB = 1
3- Write "RD_PSD_WVN_REQ" code in initiate operation register
4- Poll on OPERATION_RDY register until it becomes '1' or wait for the OPERATION_RDY interrupt pin
5- Read STATUS register to check the status of the operation
6- Read PSD_LENGTH register
7- Read the PSD from SPCTRM_DATA_OUT register successively in one frame with the number of samples known by PSD_LENGTH
8- Close the communication frame

9- Read the Wave number vector from WAVE_NUM_DATA_OUT register successively in one frame with the number of samples determined by PSD_LENGTH
10- Close the communication frame

- **Write window operation:**

1- Poll on OPERATION_RDY register until it becomes '1' or wait for the OPERATION_RDY interrupt pin
2- Write the data stream length in samples in GENERIC_DATA_IN_LEN register
3- Write AUTO_INCB = 1
4- Write "WR_WIN_REQ" code in initiate operation register
5- Poll on OPERATION_RDY register till it becomes '1' or use OPERATION_RDY interrupt pin
6- Write the window coefficients through GENERIC_DATA_IN register successively in one frame
7- Close the communication frame
8- Poll on OPERATION_RDY reg/pin = 1 indicating entered data has been checked
9- Read STATUS register to check the status of the operation if INTRPT = 1

- **Read WIN_POINTS with a read request:**

1- Poll on OPERATION_RDY register until it becomes '1' or wait for the OPERATION_RDY interrupt pin
2- Write AUTO_INCB = 1
3- Write the corresponding code in INITIATE_OPERATION register
4- Poll on OPERATION_RDY register = '1' or wait for the OPERATION_RDY interrupt pin
5- Read STATUS register to check the status of the operation if INTRPT = 1
6- Read the required data from its output register successively in one frame
7- Close the communication frame

- **Abort an ongoing operation:**

For user to abort an ongoing operation
1- Read the ABORT status bit to make sure it is cleared
2- Write "ABORT_OPERATION" register = '1'
3- Poll on "ABORT" Status bit until it becomes '1'
4- Poll on OPERATION_RDY register = '1' or wait for the OPERATION_RDY interrupt pin

Chapter 2 Mode 2: Using NeoSpectra SPI Communication Service

To enable the Raspberry PI board to communicate with NeoSpectra Micro DVK, an SPI communication service is provided on the Raspberry PI board. It is a layer implemented over SPI to provide the user with the NeoSpectra Micro set of operations. You can use this service to build your own application either on PC or on the Raspberry PI board.

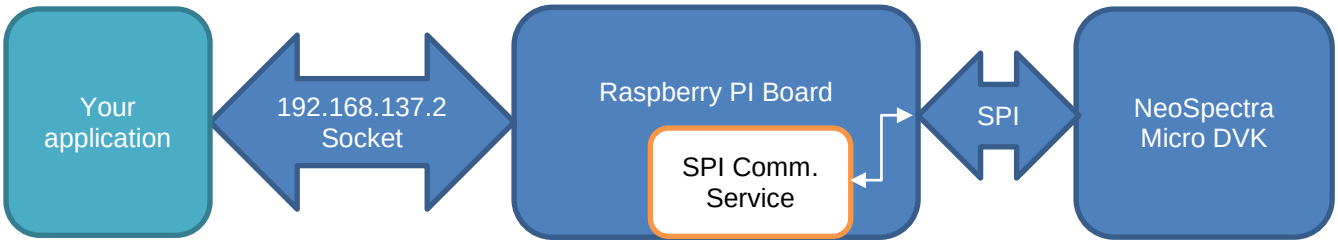


Figure 2.1: DVK Basic Block Diagram

1. Requirements

- NeoSpectra Micro DVK.
- Raspberry PI Zero W board.

2. Interface

- Any application (on raspberry PI or outside it) should communicate with NeoSpectra SPI communication service using network socket:
  The communication service IP is: **192.168.137.2**
  The read port is: **5001**
  The write port is: **5000**

3. Development package & architecture

- Program name: NSSPIService

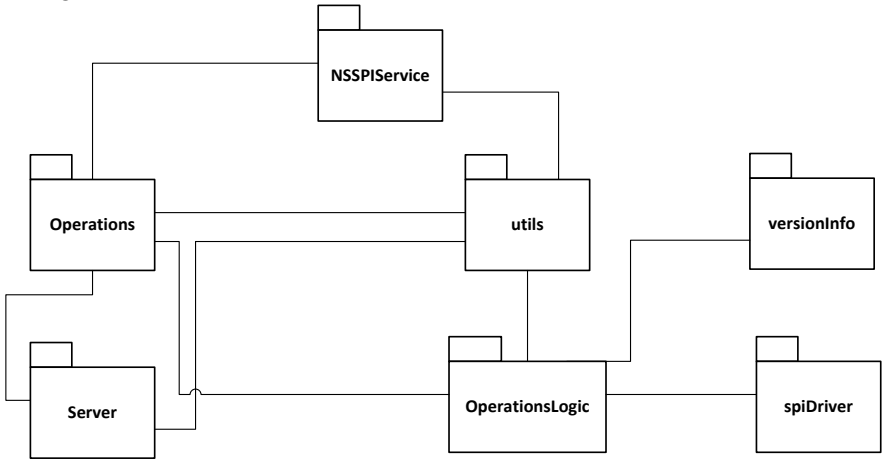


Figure 2.2: NeoSpectra SPI communication service design diagram

4. Commands

A set of operations is provided by the communication service. The same packet format should be used with all operations but only a subset of these fields is used by each function. To perform any of the predefined operations you need to do the following:

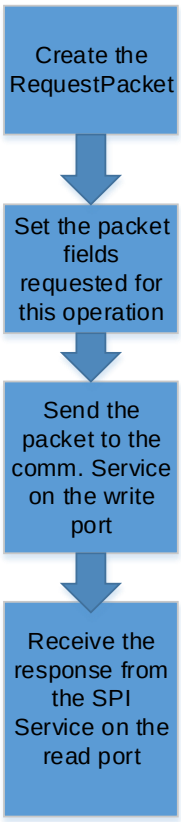


Figure 2.3: Operation Sequence

Notice that the read and write ports should be open at the beginning of the program before performing any operation.

4.1. Packet format

Packet	Data Type	Description
operation	Int	Specify the operation number requested
resolution	Int	Selects the resolution required: 0: 16nm. 1: reserved.
Mode	Int	Selects the run mode required: 0: Single mode 4: Continuous mode
zeroPadding	Int	Specify the number of points used in the FFT: 1: 8k points. 2: 16k points. 3: 32k points.

scanTime	Int	Duration of the scan in msec with a min of 10ms and a max of 2^{24}ms
commonWavNum	Int	Specifies the number of points used for the wave number: 0: Disable common wave number 1: 65 points. 2: 129 points. 3: 257 points. 4: 513 points. 5: 1024 points. 6: 2048 points. 7: 4096 points.
opticalGain	Int	0: use the optical gain settings saved on the DVK. 1: use the calculated optical gain settings. 2: use external optical gain settings.
apodizationSel	Int	Select one of the Apodization windows: 0: Boxcar 1: Gaussian 2: Happ-Genzel 3: Lorenz
GeneralData	Int[40]	The first five elements are used for specifying the wavelength of the absorption lines of a certain standard calibrator material‡ during wavelength correction routine. And also used in other routines other than the wavelength correction as general purpose fields.

4.2. Operations Description

4.2.1. General Rule

- Length of the packet should be sent at the beginning of each packet.

Length	Data

Length: 4 bytes specify the length of Data field
Data: packet fields required to be filled for every operation.

4.2.2. Operation: readModuleID

- Description: Returns the ModuleID of the connected Neospectra DVK

- Required data packet fields to be filled:

Operation	1
resolution	Value Not Required
Mode	Value Not Required
zeroPadding	Value Not Required
scanTime	Value Not Required
commonWavNum	Value Not Required
opticalGain	Value Not Required
apodizationSel	Value Not Required
GeneralData	Value Not Required

‡ The values should be quantized before passing them (multiplied by 2^{20})

- Received response packet:

Packet Field	Length	Description
ModuleID	21 bytes (Max length)	Unique identifier of DVK (null terminated string)

4.2.3. Operation: checkBoard

- Description: Check the status of the connected DVK. Returns 0 if the board is connected and initialized.

- Required packet fields to be filled:

Operation	2
resolution	Value Not Required
Mode	Value Not Required
zeroPadding	Value Not Required
scanTime	Value Not Required
commonWavNum	Value Not Required
opticalGain	Value Not Required
apodizationSel	Value Not Required
GeneralData	Value Not Required

- Received response packet:

Packet Field	Length	Description
Status	1 byte	0: No error >0: Error (Number)

4.2.4. Operation: runPSD

- Description: Requests to perform a scan and returns a Power Spectral Density (PSD)

- Required packet fields to be filled:

Operation	3
resolution	0
Mode	Required
zeroPadding	Required
scanTime	Required
commonWavNum	Required
opticalGain	Required
apodizationSel	Required
GeneralData	Value Not Required

- Received response packet:

Packet Field	Length	Description
Status	4 bytes	0: No error >0: Error (Number)
Length	4 bytes	Length of the PSD and wavenumber (Number)
PSD	8*4096 bytes	Returned PSD[§] (Array of numbers, 8

[§] Returned values are quantized. It should be divided by 2^{33} to de-quantize them.

		bytes each)
Wavenumber	8*4096 bytes	Corresponding Wavenumber values** (Array of numbers, 8 bytes each)

4.2.5. Operation: runBackground

- Description: performs a background reading

- Required packet fields to be filled:

Operation	4
resolution	0
Mode	Required
zeroPadding	Required
scanTime	Required
commonWavNum	Required
opticalGain	Required
apodizationSel	Required
GeneralData	Value Not Required

- Received response packet:

Packet Field	Length	Description
Status	1 byte	0: No error >0: Error (Number)

Operation: runAbsorbance

- Description: perform a scan and returns the absorbance.
- Prerequisite operation: runBackground

- Required packet fields to be filled:

Operation	5
resolution	0
Mode	Required
zeroPadding	Required
scanTime	Required
commonWavNum	Required
opticalGain	Required
apodizationSel	Required
GeneralData	Value Not Required

Note: Same input values as runBackground should be used

- Received response packet:

Packet Field	Length	Description
Status	4 bytes	0: No error >0: Error (Number)
Length	4 bytes	Length of the absorbance and wavenumber (Number)
Absorbance	8*4096 bytes	Returned absorbance††

** Returned values are quantized. It should be divided by 2^{30} to de-quantize them.

		(Array of numbers, 8 bytes each)
Wavenumber	8*4096 bytes	Corresponding Wavenumber values‡‡ (Array of numbers, 8 bytes each)

4.2.6. Operation: runGainAdj

- Description: Calculate the required gain for a certain sample

- Required packet fields to be filled:

Operation	6
resolution	Value Not Required
Mode	Value Not Required
zeroPadding	Value Not Required
scanTime	Value Not Required
commonWavNum	Value Not Required
opticalGain	Value Not Required
apodizationSel	Value Not Required
GeneralData	Value Not Required

- Received response packet:

Packet Field	Length	Description
Status	1 byte	0: No error >0: Error (Number)
Gain code	2 bytes	

4.2.7. Operation: BurnGain

- Description: Burns the calculated gain adjustment on the DVK
- Prerequisite Operation: runGainAdj

- Required packet fields to be filled:

Operation	7
resolution	Value Not Required
Mode	Value Not Required
zeroPadding	Value Not Required
scanTime	Value Not Required
commonWavNum	Value Not Required
opticalGain	Value Not Required
apodizationSel	Value Not Required
GeneralData	Value Not Required

- Received response packet:

Packet Field	Length	Description
Status	1 byte	0: No error >0: Error (Number)

†† Returned values are quantized. It should be divided by 2^{33} to de-quantize them
‡‡ Returned values are quantized. It should be divided by 2^{30} to de-quantize them

4.2.8. Operation: BurnSelf

- Description: Burns the self correction parameters on the DVK
- Prerequisite Operation: runSelfCorr

- Required packet fields to be filled:

Operation	8
resolution	Value Not Required
Mode	Value Not Required
zeroPadding	Value Not Required
scanTime	Value Not Required
commonWavNum	Value Not Required
opticalGain	Value Not Required
apodizationSel	Value Not Required
GeneralData	Value Not Required

- Received response packet:

Packet Field	Length	Description
Status	1 byte	0: No error >0: Error (Number)

4.2.9. Operation: BurnWLN

- Description: Burns the wavelength correction parameters on the DVK
- Prerequisite Operation: runWavelengthCorrBG, runWavelengthCorr

- Required packet fields to be filled:

Operation	9
resolution	Value Not Required
Mode	Value Not Required
zeroPadding	Value Not Required
scanTime	Value Not Required
commonWavNum	Value Not Required
opticalGain	Value Not Required
apodizationSel	Value Not Required
GeneralData	Value Not Required

- Received response packet:

Packet Field	Length	Description
Status	1 byte	0: No error >0: Error (Number)

4.2.10. Operation: runSelfCorr

- Description: Calculates the self-correction parameters

- Required packet fields to be filled:

Operation	10
resolution	0
Mode	Value Not Required
zeroPadding	Required
scanTime	Required
commonWavNum	Required

opticalGain	Required
apodizationSel	Required
GeneralData	Value Not Required

- Received response packet:

Packet Field	Length	Description
Status	1 byte	0: No error >0: Error (Number)

4.2.11. Operation: runWavelengthCorrBG

- Description: Takes a background reading for the wavelength correction

- Required packet fields to be filled:

Operation	11
resolution	0
Mode	Value Not Required
zeroPadding	Required
scanTime	Required
commonWavNum	Required
opticalGain	Required
apodizationSel	Required
GeneralData	Value Not Required

- Received response packet:

Packet Field	Length	Description
Status	1 byte	0: No error >0: Error (Number)

4.2.12. Operation: runWavelengthCorr

- Description: performs the wavelength correction
- Prerequisite Operation: runWavelengthCorrBG

- Required packet fields to be filled:

Operation	12
resolution	0
Mode	Value Not Required
zeroPadding	Required
scanTime	Required
commonWavNum	Required
opticalGain	Required
apodizationSel	Required
GeneralData	Peaks of the reference material used in wavelength correction[§§] *Note: Maximum peaks to be used (5) peaks.

Note: Same input values as runWavelengthCorrBG should be used

[§§] The values should be quantized before passing them (multiplied by 2^{20})

- Received response packet:

Packet Field	Length	Description
Status	1 byte	0: No error >0: Error (Number)

4.2.13. Operation: restoreDefault

- Description: restores the default gain and correction parameters

- Required packet fields to be filled:

Operation	13
resolution	Value Not Required
Mode	Value Not Required
zeroPadding	Value Not Required
scanTime	Value Not Required
commonWavNum	Value Not Required
opticalGain	Value Not Required
apodizationSel	Value Not Required
GeneralData	Value Not Required

- Received response packet:

Packet Field	Length	Description
Status	1 byte	0: No error >0: Error (Number)

4.2.14. Operation: readSoftwareVersion

- Description: returns the version of the software on the DVK

- Required packet fields to be filled:

Operation	14
resolution	Value Not Required
Mode	Value Not Required
zeroPadding	Value Not Required
scanTime	Value Not Required
commonWavNum	Value Not Required
opticalGain	Value Not Required
apodizationSel	Value Not Required
GeneralData	Value Not Required

- Received response packet:

Packet Field	Length	Description
DVK version	4 bytes	Version of the software on the DVK (Number)
Pi version	4 bytes	Version of the software on the Raspberry Pi board (Number)

4.2.15. Operation: SourceSettings

- Description: Send the settings of the light source.

- Required packet fields to be filled:

Operation	22
resolution	Value Not Required
Mode	Value Not Required
zeroPadding	Value Not Required
scanTime	Value Not Required
commonWavNum	Value Not Required
opticalGain	Value Not Required
apodizationSel	Value Not Required
GeneralData	GeneralData [0]: byte0 → lamps count : byte1 → selection of the lamp GeneralData [1]: byte0 → t1 : byte1 → t2 : byte2 → delta t : byte3 → 1

Notes:

1. Lamps count defines if you want to use the two lamps in the light source or just one lamp (Possible values: 1, 2).
2. Selection of the lamp defines which lamp you want to use in case you selected lamps count = 1. (Possible values: 0, 1).
3. T1 defines delay time after opening the source in 50 ms unit. (Possible values: Any integer number <= 255).
4. T2 defines delay time before closing the source in 50 ms unit. (Possible values: Any integer number <= 255).
5. Delta t defines Delay time between opening/closing the two lamps of the source in 50 ms unit. (Possible values: Any integer number <= 255).

- Received response packet:

Packet Field	Length	Description
Status	1 byte	0: No error >0: Error (Number)

4.2.16. Operation: setOpticalSettings

- Description: Select the optical gain settings to be used during the scan.

- Required packet fields to be filled:

Operation	27
resolution	Value Not Required
Mode	Value Not Required
zeroPadding	Value Not Required
scanTime	Value Not Required
commonWavNum	Value Not Required
opticalGain	Value Not Required
apodizationSel	Value Not Required
GeneralData	GeneralData [0]: gain value *if gain value is zero, the default gain settings which are burned on Flash will be used.

- Received response packet:

Packet Field	Length	Description
Status	1 byte	0: No error >0: Error (Number)

4.2.17. Operation: injectExternalWindow

- Description: Inject external apodization window coefficients (20 coefficients maximum).

- Required packet fields to be filled:

Operation	28
resolution	Value Not Required
Mode	Value Not Required
zeroPadding	Value Not Required
scanTime	Value Not Required
commonWavNum	Value Not Required
opticalGain	Value Not Required
apodizationSel	Value Not Required
GeneralData	GeneralData [0]: least 32 bit of coefficient0 GeneralData [1]: most 32 bit of coefficient0 GeneralData [2]: least 32 bit of coefficient1 GeneralData [3]: most 32 bit of coefficient1 GeneralData [4]: least 32 bit of coefficient2 GeneralData [5]: most 32 bit of coefficient2 …

*Note:
The apodization window coefficients must be quantized by the following fraction lengths:

Coefficient	Quantization fraction length
Coefficient 0	63
Coefficient 1	59
Coefficient 2	57
Coefficient 3	54
Coefficient 4	53
Coefficient 5	53
Coefficient 6	53
Coefficient 7	54
Coefficient 8	54
Coefficient 9	55
Coefficient 10	56
Coefficient 11	54
Coefficient 12	54
Coefficient 13	54
Coefficient 14	56
Coefficient 15	58
Coefficient 16	60
Coefficient 17	59
Coefficient 18	62
Coefficient 19	62

- Received response packet:

Packet Field	Length	Description
Status	1 byte	0: No error >0: Error (Number)

Chapter 3 SDK

1. Installation

SpectroMOST Micro should be installed before proceeding with the SDK installation steps.
After downloading the SDK package the following steps should be performed in Eclipse IDE:

1.1. Opening Project:

Apply the following steps:

1. Click File → New → Project → Java Project.
2. Brows to your SDK folder location.
3. In source tab:
   - Make sure that you've 3 folders marked as source folders (p3AppManager_micro/src, spectromost_micro/src, release)
   - In case not all of the previous folders were marked as source folders, right click on that folder and select "Use as source folder".
   - Ensure that the "Default output folder" field contains the path to the bin folder.
4. Press finish.

1.2. Run configuration:

In the run configuration window apply the following steps:

1. Java Application → new configuration.
2. In main tab: main class → search for(Userinterface).
3. In argument tab :
   - VM arguments: write the following command:
     -Djava.library.path="bin_path_inside_SDK_folder"
     -Dswing.defaultlaf=com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel
   - Working directory→ ${workspace_loc:SDK_MOSTAPP/bin}

2. Software Architecture

SpectroMOST Micro application has the components described below.

1. Application software

- spectromost.jar: The source code of SpectroMOST Basic Edition is delivered as for reference. This component should be replaced by the end-use application software.
- 3rd party modules used by spectromost.jar:
  - jcommon-1.0.21.jar
  - jfreechart-1.0.17.jar
  - log4j-1.2.17.jar
  - miglayout15-swing.jar

2. Spectrometer driver:

- p3AppManager_micro.jar (which is the only component from which spectromost.jar calls the different APIs)

3. APIs

p3AppManager_micro APIs

The p3AppManager component has the following APIs:

1. **Interface: p3AppManagerImpl()**
Description: Component Constructor

Inputs	Outputs	Return	Type
String dir (optional): Set the working directory of the SDK.	-	-	Sync

2. **Interface: addObserver()**
Description: Add the caller as an observer in the p3AppManager

Inputs	Outputs	Return	Type
Reference to the caller instance.	-	-	Sync

Notes:

- Guidelines to get the status of the software:

- Your class should implement "Observer" interface.
- The class should add itself as an observer to "p3AppManager" class through addObserver() method.
- Update() method will be invoked from p3AppManager once an action has been finished. This method should be overridden also in your class.

3. **Interface: getDeviceId()**
Description: Gets the ID of the connected spectrometer module.

Inputs	Outputs	Return	Type
-	String deviceID	Spectrometer ID	Sync

4. **Interface: initializeCore()**
Description: Begin initializing the connected board

Inputs	Outputs	Return	Type
-	-	p3AppManagerStatus: See Table 3	Async

5. **Interface: runSpec()**
Description: Generate Spectrum (relative to background measurement)

Inputs	Outputs	Return	Type
- String runTime: Scan time in milliseconds - isSample: false means	-	p3AppManagerStatus: See Table 3	Async

background and true means sample - String apodization (optional) - String zeroPadding (optional) - String gainValue - String NumberOfDataPoints See Table 1 - String continues mode: Set by 1 if continues run is taken and set by zero if single run is taken			

6. **Interface: getSpecData()**
Description: Get data corresponding to runSpec function

Inputs	Outputs	Return	Type
-	See Table 2	double[][]	Sync

7. **Interface: runInterSpec()**
Description: Generate Interferogram and Power Spectral Density

Inputs	Outputs	Return	Type
- String runTime: Scan time in milliseconds - String apodization (optional) - String zeroPadding (optional) - String gainValue - String NumberOfDataPoints See Table 1 - String continues mode: Set by 1 if continues run is taken and set by zero if single run is taken	-	p3AppManagerStatus: See Table 3	Async

8. **Interface: getInterSpecData()**
Description: Get data corresponding to runInterSpec command

Inputs	Outputs	Return	Type
-	See Table 2	double[][]	Sync

9. **Interface: checkDeviceStatus()**
Description: Check the current status of the connected device

Inputs	Outputs	Return	Type
-	-	p3AppManagerStatus:	Sync

		See Table 3	

10. **Interface: wavelengthCalibrationBG()**
Description: Perform first step of the wavelength calibration using background reading

Inputs	Outputs	Return	Type
- String runTime: Scan time in milliseconds - String apodization (optional) - String zeroPadding (optional) See Table 1 - String gainValue - String NumberOfDataPoints	-	p3AppManagerStatus: See Table 3	Async

11. **Interface: wavelengthCalibration()**
Description: Perform second step of the wavelength calibration using a known calibrator (sample)

Inputs	Outputs	Return	Type
- String runTime: Scan time in milliseconds - String calibrator Type: name of the sample to be used - String apodization (optional) - String zeroPadding (optional) See Table 1 - String gainValue - String NumberOfDataPoints	-	p3AppManagerStatus: See Table 3	Async

12. **Interface: runSpecGainAdjBG()**
Description: Add a new gain for the spectrum using background

Inputs	Outputs	Return	Type
- String runTime: Scan time in milliseconds	-	p3AppManagerStatus: See Table 3	Async

13. **Interface: getGainAdjustSpecData()**
Description: Get gain settings corresponding to runSpecGainAdjBG()

Inputs	Outputs	Return	Type
-	-	double[][]	Sync

14. **Interface: burnSpecificSettings()**
Description: Burn specific gain settings and enable/disable the saving of the wavenumber correction values on the module

Inputs	Outputs	Return	Type
- String [] settingsToBurn: List containing the name of the gain settings to burn - String updateCorrection: flag if set to true it saves the correction values to the module.	-	p3AppManagerStatus: See Table 3	Async

15. **Interface: restoreDefaultSettings()**
Description: Restore the default gain settings and wavenumber correction settings from the module

Inputs	Outputs	Return	Type
-	-	p3AppManagerStatus: See Table 3	Async

16. **Interface: setWorkingDirectory()**
Description: Sets the working directory of the application

Inputs	Outputs	Return	Type
- String dir: Path to the working directory	-	-	Async

17. **Interface: getWorkingDirectory()**
Description: return the current working directory of the application

Inputs	Outputs	Return	Type
-	-	- String : Path to the working directory	Async

18. **Interface: setExternalApodizationWindow()**
Description: Sets the Apodization window with an external window from the user.

Inputs	Outputs	Return	Type
-Long[] apodizationWindow: External window defined by user	-	-	Async

19. **Interface: getSoftwareVersion()**
Description: Return the software version number

Inputs	Outputs	Return	Type
-	-	- String : Software version number	Async

Input Data Format

Parameter	Description	Value	Description
Apodization	Shape of the window to be used to multiply the Interferogram before FFT	Boxcar	
		Gaussian	
		Happ-Genzel	
		Lorenz	
ZeroPadding	Number of points to be added to the Interferogram before FFT	0	No points to add
		1	1*VALUE= number of points to add
		3	3*VALUE= number of points to add
OpticalGainPrefix	Identifier between Interferogram gain settings and Spectrum gain settings	_InterSpec_	To retrieve the gain in case of background or interferogram
		Spec	To retrieve the gain in case of Sample
NumberOfDataPoints		65 pts	
		129 pts	
		257 pts	
		513 pts	
		1024 pts	
		2048 pts	
		4096 pts	

Table 1: Input data format

Output Data Format

Two-dimensional array holds the spectrum/interferogram data which consists of the following arrays:

API Name	Array Index	Description	Data Set	Axis	Units
getInterSpecData()	0	Optical path difference values	Interferogram	X	µm
	1	Photo detector's current intensity values (Interference pattern)	Interferogram	Y	nA
	2	Wavenumber values	Spectrum	X	cm-1

	3	Power spectral density (PSD) values	Spectrum	Y	a.u.
getSpecData()	2	Wavenumber values	Spectrum	X	cm-1
	3	Absorbance values (relative to background measurement)	Spectrum	Y	Abs.

Table 2: Input data format

p3AppManagerStatus

Status Code	Enum	Message
0	*NO_ERROR*	No error
1	*DEVICE_BUSY_ERROR*	Device is busy.
2	*BOARD_DISTCONNECTED_ERROR*	SpectroMOST does not detect any connected NeoSpectra module
3	*BOARD_NOT_INITIALIZED_ERROR*	NeoSpectra module is not initialized
4	*UNKNOWN_ERROR*	Unknown error. Contact Si-Ware Systems
7	*CONFIG_FILES_LOADING_ERROR*	Error in loading resolution folder
8	*CONFIG_PARAM_LENGTH_ERROR*	Error in resolution folder format
11	*INVALID_RUN_TIME_ERROR*	Invalid scan time
23	*INAVLID_REG_FILE_FORMAT_ERROR*	Error in resolution folder format
24	*NO_OF_SCANS_DSP_ERROR*	DSP error
25	*DSP_INTERFEROGRAM_POST_PROCESSINF_ERROR*	DSP error
26	*DSP_INTERFEROGRAM_POST_EMPTY_DATA_ERROR*	DSP error
27	*DSP_INTERFEROGRAM_POST_BAD_DATA_ERROR*	DSP error
28	*UPDATE_CORR_FILE_ERROR*	Error updating resolution folder
29	*WHITE_LIGHT_PROCESSING_ERROR*	Error in saving background data
30	*DSP_INTERFEROGRAM_FFT_POST_PROCESSINF_ERROR*	DSP error
31	*INVALID_RUN_PARAMETERS_ERROR*	Invalid run parameters
32	*INVALID_RUN_TIME_NOT_EQUAL_BG_RUN_TIME_ERROR*	Background measurement scan

		time is not equal to sample measurement scan time
33	*NO_VALID_BG_DATA_ERROR*	No valid background measurement found
34	*INTERFERO_FILE_CREATION_ERROR*	Error occurred during saving interferogram data
35	*PSD_FILE_CREATION_ERROR*	Error occurred during saving PSD data
36	*SPECTRUM_FILE_CREATION_ERROR*	Error occurred during saving spectrum data
37	*GRAPHS_FOLDER_CREATION_ERROR*	Error occurred during creating data folder
38	*INVALID_APODIZATION_WINDOW*	Error occurred while loading an invalid apodization window number
42	*INITIATE_MIPDRIVER_ERROR*	Error occurred during NeoSpectra module initialization
43	*INVALID_BOARD_CONFIGURATION_ERROR*	Error occurred during NeoSpectra module initialization
50	*DATA_STREAMING_TAIF_ERROR*	Error occurred during streaming from NeoSpectra module
51	*DATA_STREAMING_ERROR*	Error occurred during streaming from NeoSpectra module
52	*INVALID_NOTIFICATION_ERROR*	Error occurred during result return
53	*INVALID_ACTION_ERROR*	Invalid action performed
54	*INVALID_DEVICE_ERROR*	Invalid device is attached
55	*THREADING_ERROR*	Threading error occurred
60	*ACTUATION_SETTING_ERROR*	Error occurred during the setup of actuation settings
61	*DEVICE_IS_TURNED_OFF_ERROR*	NeoSpectra module is switched off
62	*ASIC_REGISTER_WRITING_ERROR*	Error occurred during writing to chip registers
110	*FAILED_IN_ADAPTIVE_GAIN*	Error occurred while save gain settings
111	*ASIC_REGISTER_READING_ERROR*	Error occurred during ASIC register reading
116	*WAVELENGTH_CALIBRATION_ERROR*	Calibrator has no wavelengths in the detector range
117	*NO_VALID_OLD_MEASUREMENT_ERROR*	Error occurred while there is no old

		measurement found
118	*DSP_UPDATE_FFT_SETTINGS_ERROR*	Error while make DSP data update FFT settings
199	*USBCommunicationTimeOutError*	Error occurred during USB communication
201	*CommunicationWriteError*	Error occurred during TAIF writing register
202	*CommunicationReadError*	Error occurred during TAIF reading register
203	*FLASHING_CONFIGURATION_ERROR*	Error occurred during flash the program
213	*ROM_INVALID_ID*	sample ID isn't correct
214	*DEVICE_NOT_INITIALIZED_ERROR*	Error occurred if device is not initialized
218	*SAMPLE_FOLDERS_INVALID_ERROR*	Error occurred if sample folder is not supported
228	*OPTICAL_FILE_ERROR*	Error occurred during optical sittings
229	*NOT_ENOUGH_MEMORY_ERROR*	Not enough memory error
230	*I2_STAT_INT1_END_TIMEOUT*	ASIC returned error during interpolation from block1
231	*I2_STAT_INT1_END_INVALID*	ASIC returned error during interpolation from block1
232	*I2_STAT_INT1_AVG_OVERFLOW*	ASIC returned error during interpolation from block1
233	*I2_STAT_INT1_CORE_INVALID_REGION*	ASIC returned error during interpolation from block1
234	*I2_STAT_INT1_CORE_TIMEOUT*	ASIC returned error during interpolation from block1
235	*I2_STAT_INT1_CORE_OVERFLOW*	ASIC returned error during interpolation from block1
236	*I2_STAT_INT1_START_TIMEOUT*	ASIC returned error during interpolation from block1
237	*I2_STAT_INT2_END_TIMEOUT*	ASIC returned error during interpolation from block2
238	*I2_STAT_INT2_END_INVALID*	ASIC returned error during interpolation from block2
239	*I2_STAT_INT2_AVG_OVERFLOW*	ASIC returned error during interpolation from block2
240	*I2_STAT_INT2_CORE_INVALID_REGION*	ASIC returned error

		during interpolation from block2
241	*I2_STAT_INT2_CORE_TIMEOUT*	ASIC returned error during interpolation from block2
242	*I2_STAT_INT2_CORE_OVERFLOW*	ASIC returned error during interpolation from block2
243	*I2_STAT_INT2_START_TIMEOUT*	ASIC returned error during interpolation from block2
244	*INVALID_SAMPLE_FOLDER_VERSION*	Version number of sample folder isn't supported
245	*TAIF_STREAMING_ERROR_INT1*	
246	*STREAMING_TIMEOUT_ERROR*	Error due to timeout of the streaming interpolation data
247	*TAIF_STREAMING_ERROR_INT2*	
248	*P3_FFT_ADDRESS_ERROR*	Error occurred during reading FFT address memory
300	*FFT_WRONG_NUMBER_POINTS*	FFT number of points is not supported
249	*CRC_NOT_MATCHED*	Error occurred during check the program correctness
250	*PATTERN_NOT_MATCHED*	Error occurred during pattern is not matched
251	*FLASH_FAILED*	Error occurred while writing on flash, no more pages in flash memory
252	*IN_ADDRESS_ERROR*	Error occurred in flash address
253	*RX_OR_ERROR*	Error occurred in Flash SPI slave block
254	*WRITE_ENABLE_FAILED*	Write enable command to flash is failed
255	*WRITE_DISABLE_FAILED*	Write disable command to flash failed
256	*FLASH_BUSY_ERROR*	Flash is not responding
259	*P3_SPI_TAIF_ADDRESS_ERROR*	Error in TAIF Register address to be written or read
204	*P3_SPI_TAIF_RX_OR_ERROR*	Receive overrun flag (asserted when new operation is started before the previous data received from single access operation is read,

		cleared by reading this register)
250	*P3_SPI_TAIF_IN_ADDR_ERROR*	Memory Address pointer is out of accepted range
260	*P3_FIR_ADDRESS_ERROR*	Invalid address
261	*P3_FIR_INVALID_ADD_DATA_ERROR*	Error flag when addresses of input data and output data are not in range of assigned memory for filter 1--> invalid
262	*P3_FIR_INVALID_SAMPLES_NUMBER_ERROR*	Error flag when number of samples less than number of taps, operation will not start until number of samples >= number of taps, 1--> invalid
263	*P3_FIR_INVALID_ADD_COEFF_ERROR*	Error flag when addresses of coeff are not in range of assigned memory for filter 1--> invalid
264	*P3_FIR_ACC1_SAT_ERROR*	Saturation flag for accumulator 1 , 1 → Saturation
265	*P3_FIR_ACC2_SAT_ERROR*	Saturation flag for accumulator 2 , 1 → Saturation
266	*P3_FIR_ACC3_SAT_ERROR*	Saturation flag for accumulator 3 , 1 → Saturation
267	*P3_FIR_ACC4_SAT_ERROR*	Saturation flag for accumulator 4 , 1 → Saturation
268	*P3_LIN_INTRP_XNEW_ACC_SAT_ERROR*	Error indicates the saturation of the accumulated Xnew generated internally
269	*P3_LIN_INTRP_XNEW_THRES_SAT_ERROR*	Error indicates the saturation of Xnew generated internally as being equal to or exceeding the saturation threshold
270	*P3_LIN_INTRP_XNEW_LD_MEM_NON_MON_ERROR*	Error indicates that the Xnew loaded from memory isn't increasing/decreasing in a monotonic way
271	*P3_LIN_INTRP_XNEW_OUT_STRTXOLD_RNG_ERROR*	Error indicates that Xold(i)>Xnew and Xold(i+1)>Xnew
272	*P3_LIN_INTRP_XNEW_OUT_FNLXOLD_RANGE_E*	Error indicates that

	RROR	no more Xold data to be loaded while Xold(i)<Xnew and Xold(i+1)<Xnew
273	*P3_LIN_INTRP_XOLD_NON_MONO_ERROR*	Error Indicates that Xold isn't increasing/decreasing in a monotonic way
274	*P3_LIN_INTRP_ZERO_DIV_ZERO_ERROR*	Error indicates dividing zero by zero which means Xold(i+1)=Xold(i) = xnew
275	*P3_LIN_INTRP_SCALR_DIV_ZERO_ERROR*	Error indicates divide by zero in scalar division mode
276	*P3_LIN_INTRP_WR_XNEW_ERR_ERROR*	Error indicates Flag xnew is gated from being written to the memory as its length exceeds 32 bit
277	*P3_LIN_INTRP_DMA_ADDR_WRD_ALGN_ERROR*	Error indicates that one of the given addresses isn't word aligned (the least 2 LSB /= 0)
278	*P3_LIN_INTRP_DMA_ADDR_LSB_IN_RNG_ERRO R*	Error Indicates LSB of one of given addresses is out of the given address space for the HW Accelerator(greater than or equal x5800)
279	*P3_LIN_INTRP_DMA_ADDR_MSB_IN_RNG_ERRO R*	Error indicates MSB of one of given addresses is out of the given address space for the HW Accelerator (not equal x200)
280	*ACTION_ABORTED*	Error occurred during ISR abort operation
281	*USERINTERFACE_DMA_WRITE_ERROR*	Error occurred during DMA write operation
282	*USERINTERFACE_WRONG_OPERATION*	Error occurred during read a wrong operation
283	*WDT_WRITE_LOCK_FAILED*	Error occurred during write lock
284	*WDT_WRITE_UNLOCK_FAILED*	Error occurred during write unlock
285	*DSP_INITIALIZATION_CONFIGURATION_FILES_IS _EMPTY_ERROR*	Error occurred during DSP missing configuration data
286	*DSP_INITIALIZATION_CONFIGURATION_FILES_L ENGTH_NOT_VALID_ERROR*	Error occurred during DSP initialization

		configuration length is not valid
287	*DSP_INITIALIZATION_INVALID_INTERFEROGRAM_TYPE_ERROR*	Error occurred during DSP initialization for invalid interferogram type
288	*DSP_INTERPOLATION_LINEAR_INPUT_SIZE_ZERO_ERROR*	Error occurred during DSP interpolation step streaming input size is zero
289	*DSP_INTERPOLATION_LINEAR_OUTPUT_SIZE_ZERO_ERROR*	Error occurred during DSP interpolation step streaming output size is zero
290	*DSP_INTERPOLATION_LINEAR_DIVISION_BY_ZERO_ERROR*	Error occurred during DSP interpolation step division by ZERO
291	*DSP_MATH_DIVISION_BY_ZERO_ERROR*	Error occurred during DSP mathematical division by ZERO operation
292	*DSP_Spline_NO_POINTS_ERROR*	Error occurred during DSP spline function no of points is not correct
293	*DSP_SPLINE_KNOTS_DECREASING_ERROR*	Error during DSP Spline cubic operation
294	*DSP_SPLINE_UNKNOWN_ERROR*	Error occurred during DSP spline for unknown reason
295	*DSP_FFT_NO_POINTS_ERROR*	Error occurred during DSP FFT number of points is not correct
296	*DSP_NOISE_LEVEL_ERROR*	Error occurred during DSP noise level problem

Table 3: p3AppManagerStatus values

4. Sequence diagrams

4.1. Initialization

The initialization scenario should be run at least once for the connected NeoSpectra module. The scenario consists of the following steps:

1. Construct the p3AppManager.jar through calling p3AppManagerImpl()
2. Add your class as an observer to be notified by the p3AppManager when asking for an asynchronous action
3. Board initialization through calling InitializeCore()
4. Waiting for finishing initialization
5. Your class will be notified when module initialization is finished

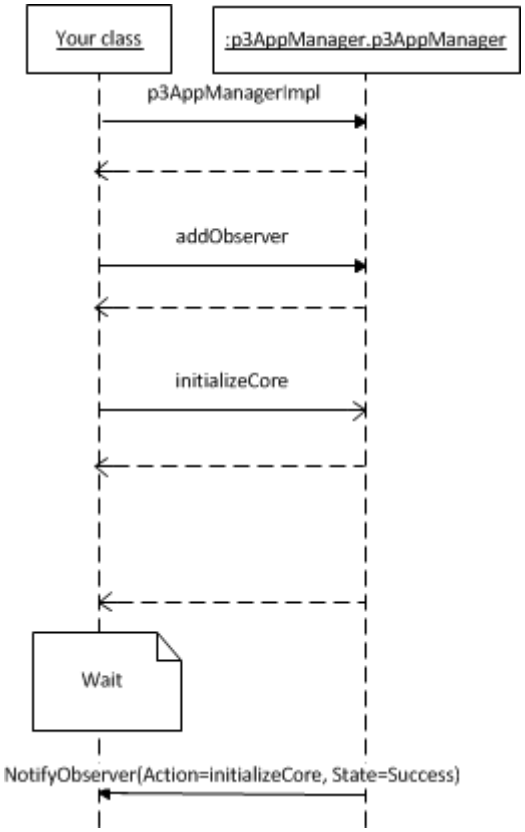


Figure 1: Initialization Sequence

4.2. Interferogram & PSD Run

The Interferogram & PSDscenario consists of the following steps:

1. Start the run procedure through calling runInterSpec(RunTime)
2. Waiting for finishing run
3. Your class will be notified when the run is finished
4. Getting the data through calling getInterSpecData()

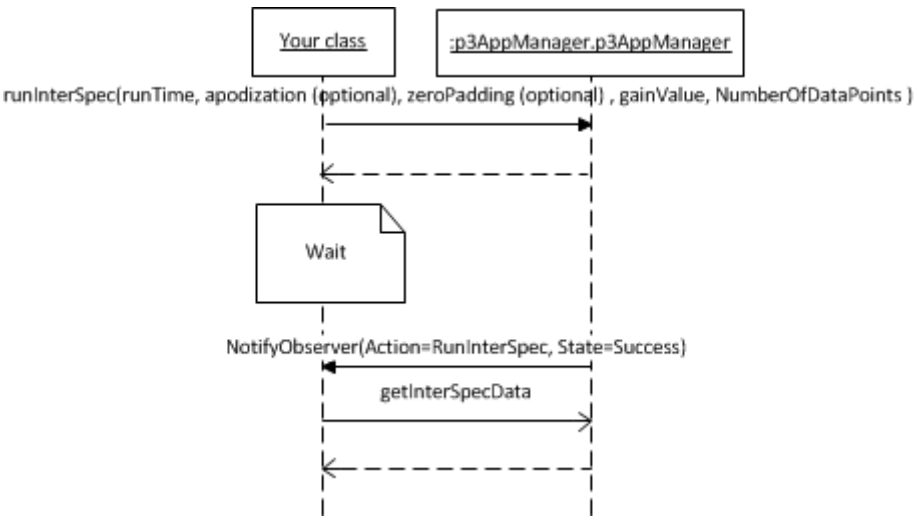


Figure : Interferogram & PSD Run Sequence

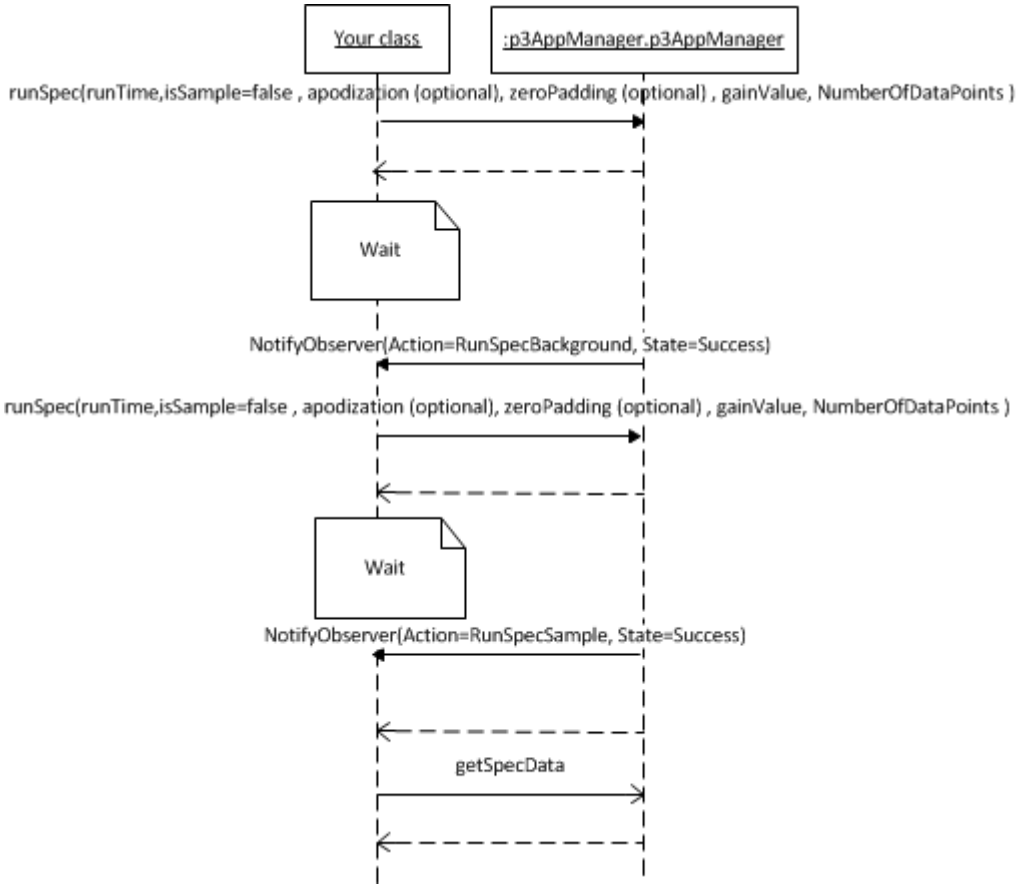


Figure 3: Spectrum Run Sequence

4.3. Spectrum Run

The Spectrum scenario consists of the following steps:

1. Start the background run procedure through calling runSpec(RunTime, isSample=false)
2. Waiting for finishing background run
3. Your class will be notified when the background run is finished
4. Start the sample run procedure through calling runSpec(RunTime, isSample=true)
5. Waiting for finishing sample run
6. Your class will be notified when the sample run is finished
7. Getting the data through calling getSpecData()

4.4. Adding Gain Settings for the Interferogram and Spectrum

Adding new gain settings for the Interferogram/ Spectrum consists of the following steps:

1. Start adjusting the gain using background by calling runSpecGainAdjBG (RunTime)
2. Waiting for finishing background run
3. Your class will be notified when the background run is finished
4. Get the new gain settings by calling getGainAdjustSpecData ()
5. To restore the default gain settings from the module, call the function restoreDefaultSettings()

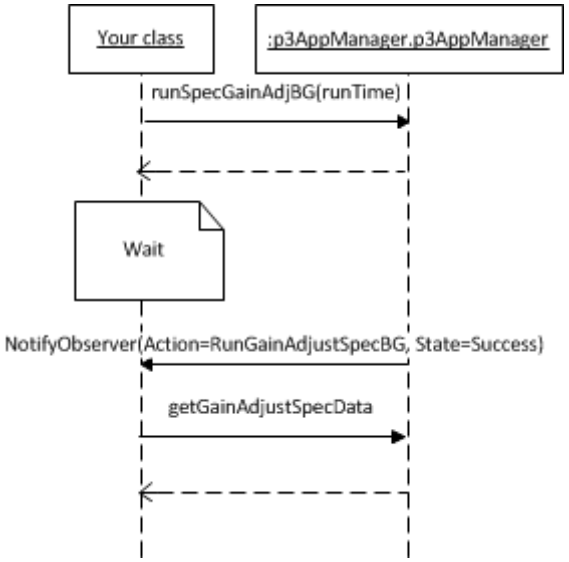


Figure 4: Interferogram Gain Adjustment

4.5. Perform Correction

Correction can be done using one of two techniques:

4.5.1. Perform Self-Correction

1. Start the correction using runCalibCorr() with a background reading
2. Wait for finishing background run

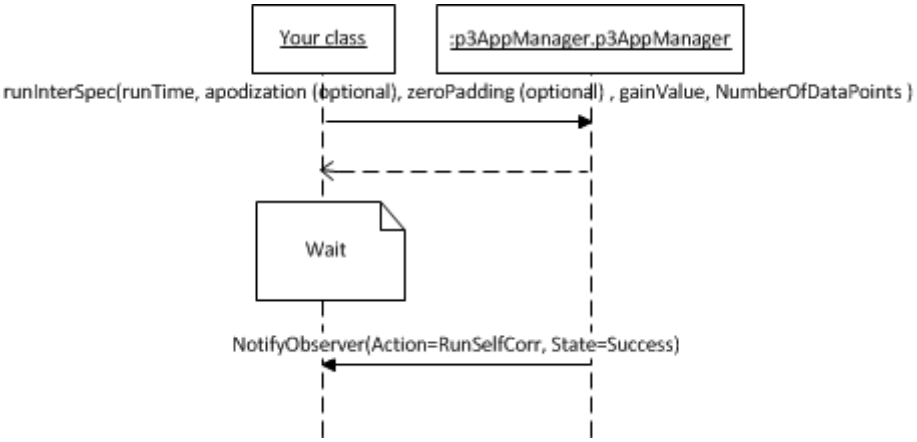


Figure 4: Self Correction

4.5.1. Perform Correction Using a Standard Sample

1. Start the first step of correction using wavelengthCalibrationBG() with a background reading
2. Wait for finishing background run
3. Start the second step of the correction using wavelengthCalibration() with a sample reading
4. Wait for finishing the sample run

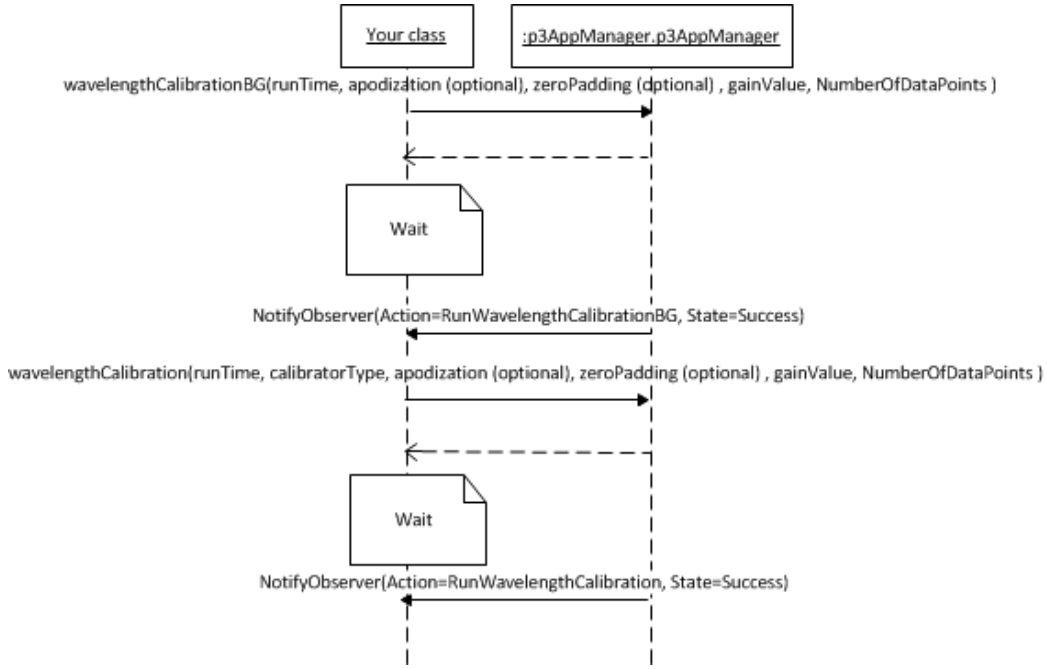


Figure 5: Correction Using Standard Sample