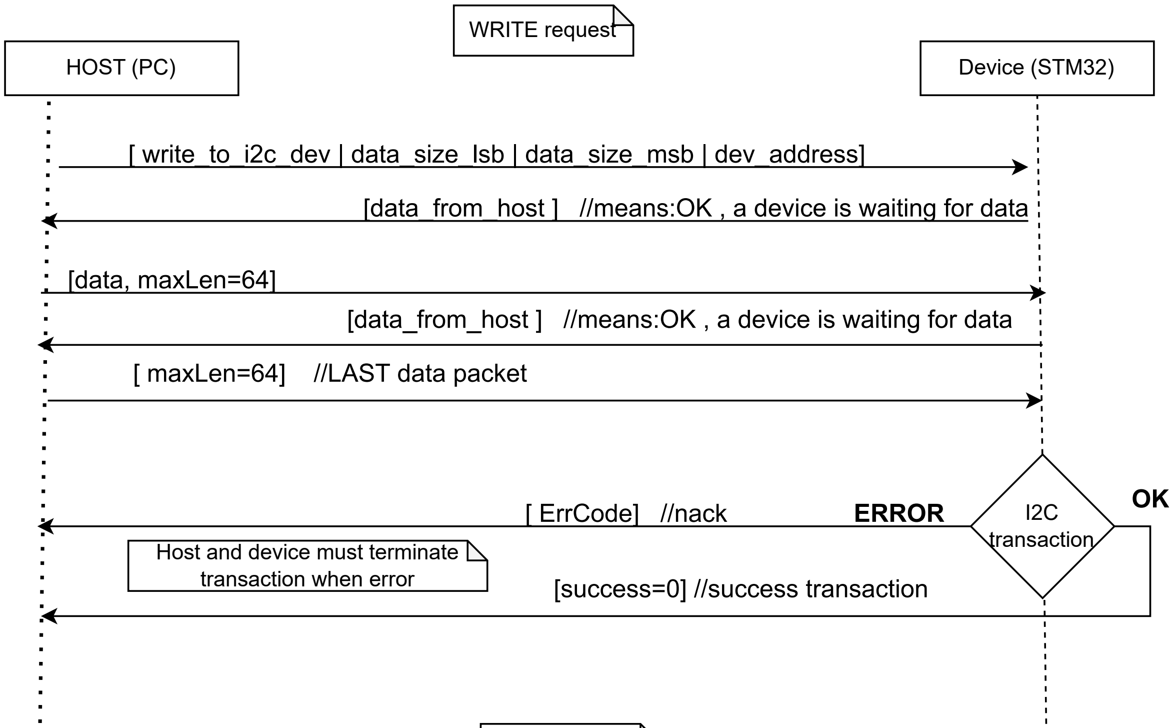
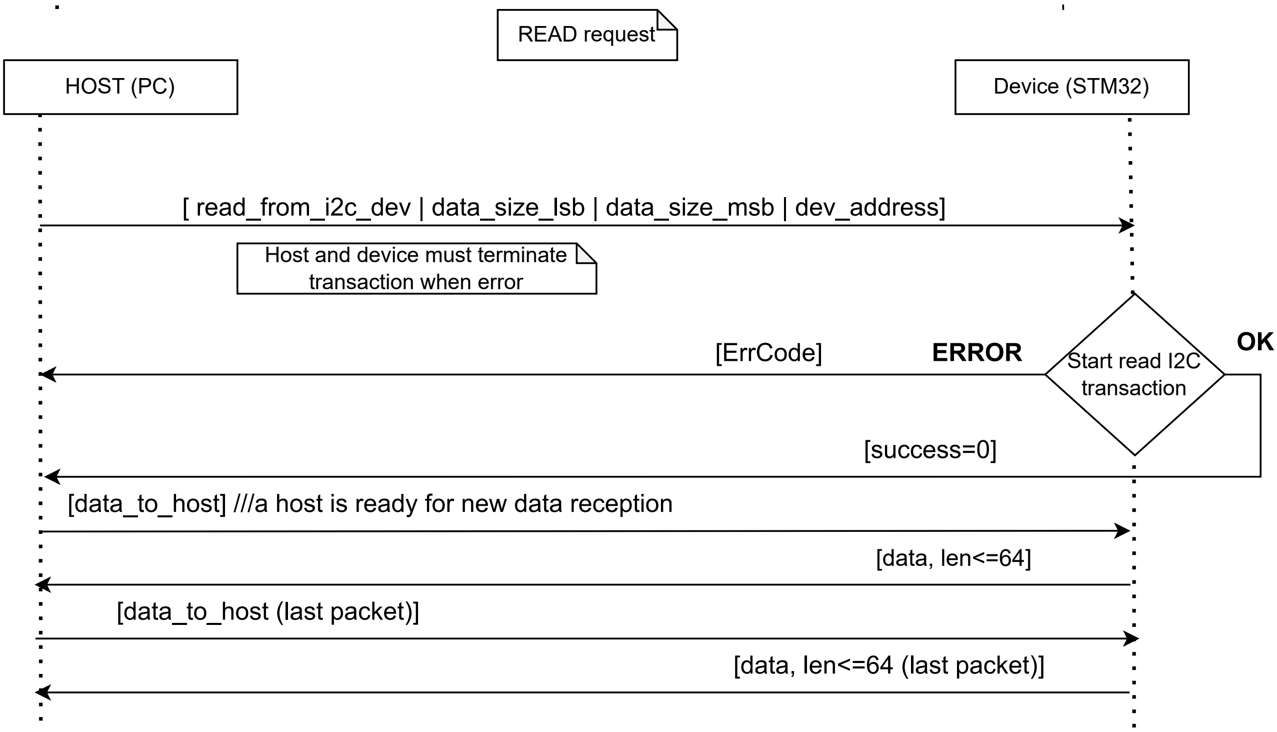
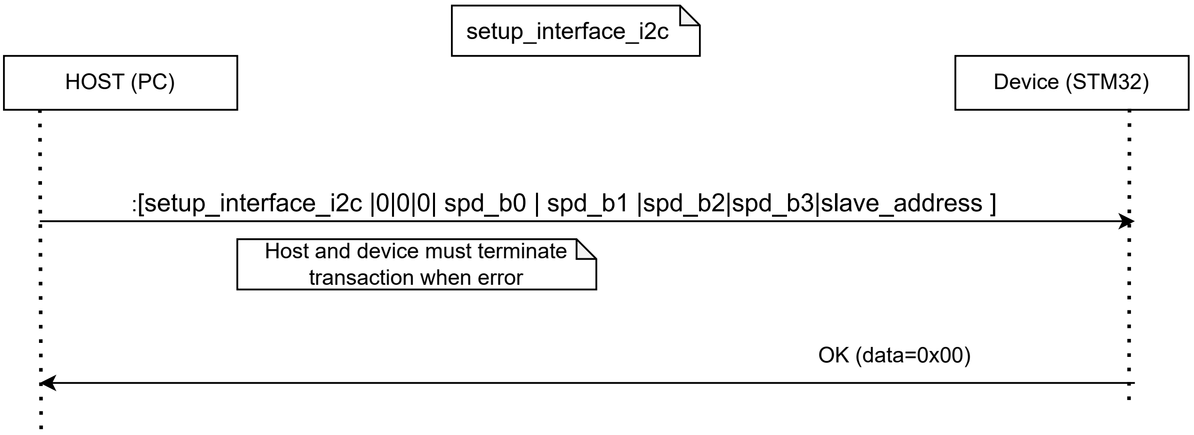
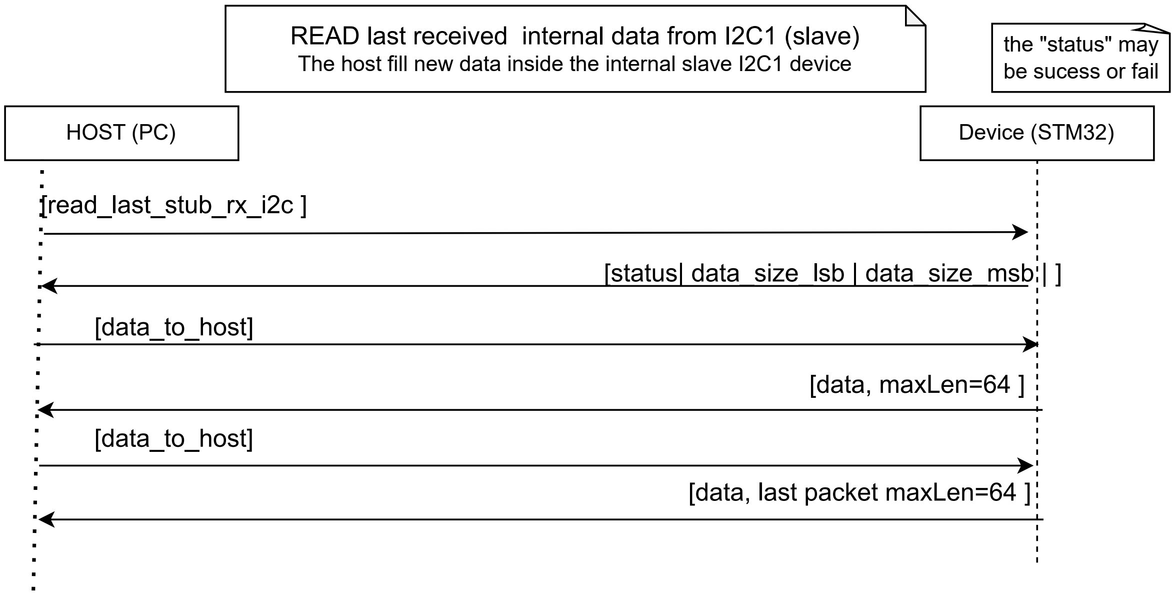
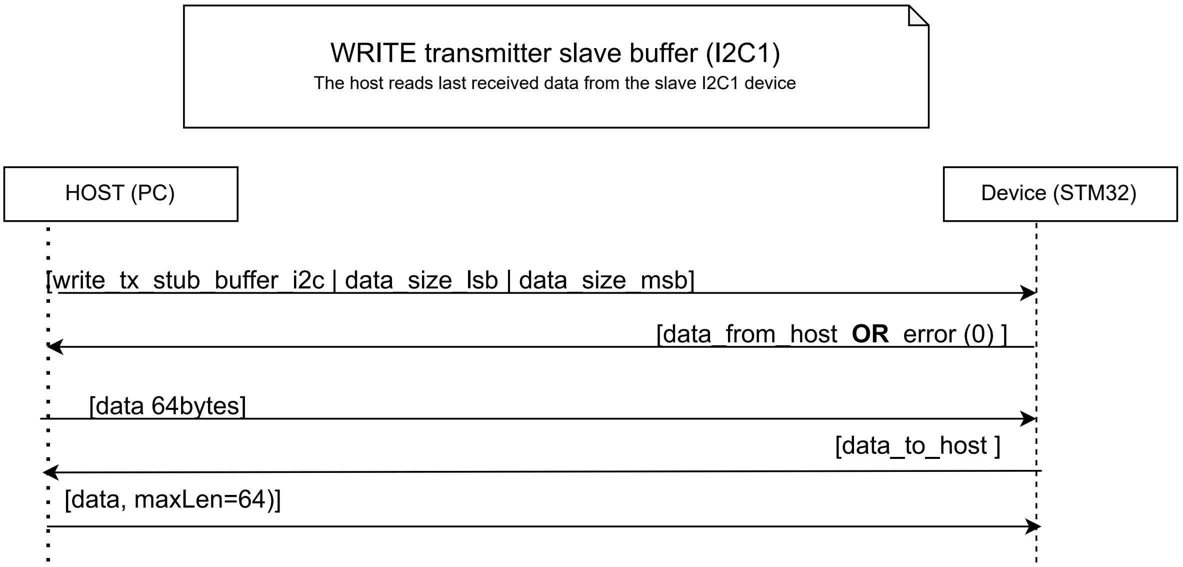
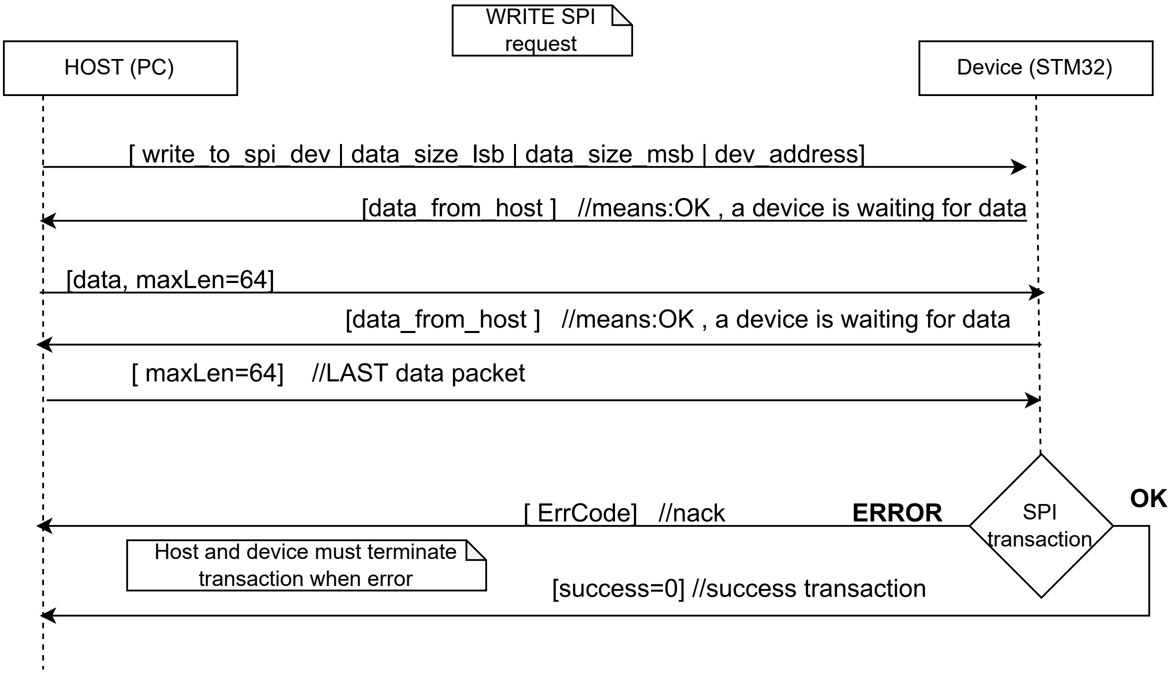

WRITE request
HOST (PC)
Device (STM32)
[write_to_i2c_dev | data_size_lsb | data_size_msb | dev_address]
[data_from_host]   //means:OK , a device is waiting for data
[data, maxLen=64]
[data_from_host]   //means:OK , a device is waiting for data
[maxLen=64]   //LAST data packet
[ErrCode]   //nack
ERROR
I2C transaction
OK
Host and device must terminate transaction when error
[success=0] //success transaction

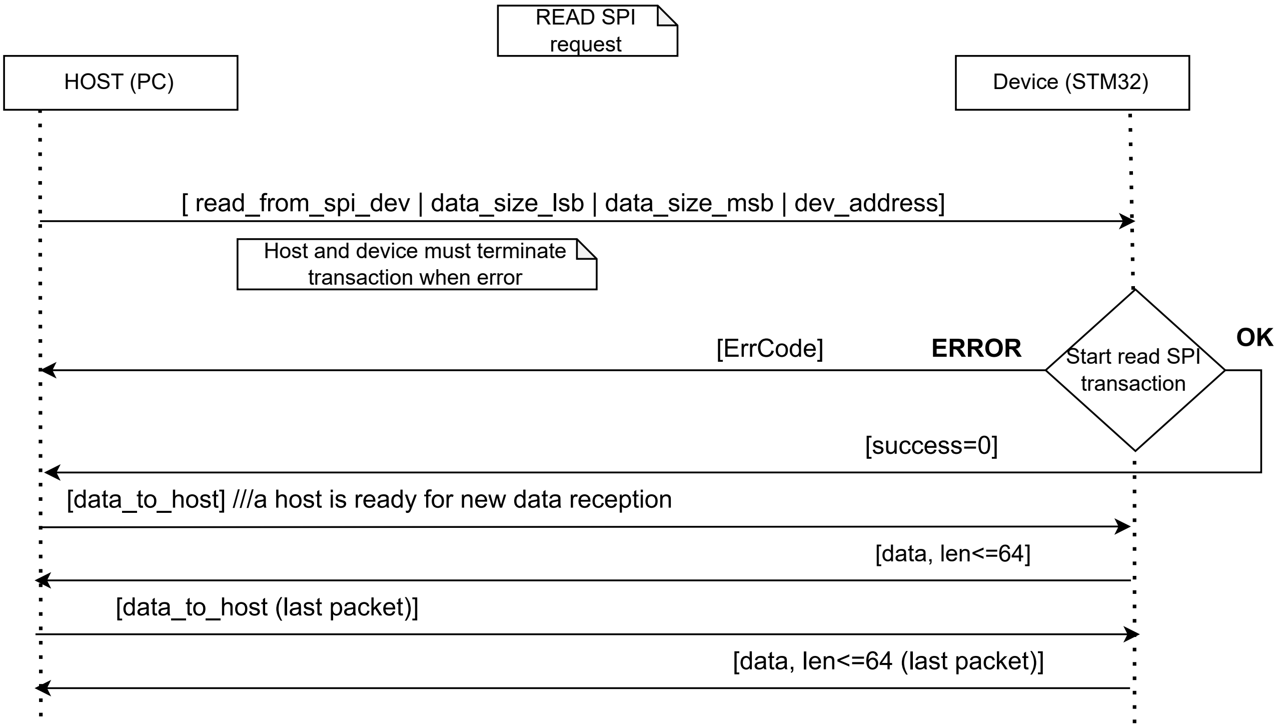

READ request
HOST (PC)
Device (STM32)
[read_from_i2c_dev | data_size_lsb | data_size_msb | dev_address]
Host and device must terminate transaction when error
[ErrCode]
ERROR
Start read I2C transaction
OK
[success=0]
[data_to_host] ///a host is ready for new data reception
[data, len<=64]
[data_to_host (last packet)]
[data, len<=64 (last packet)]

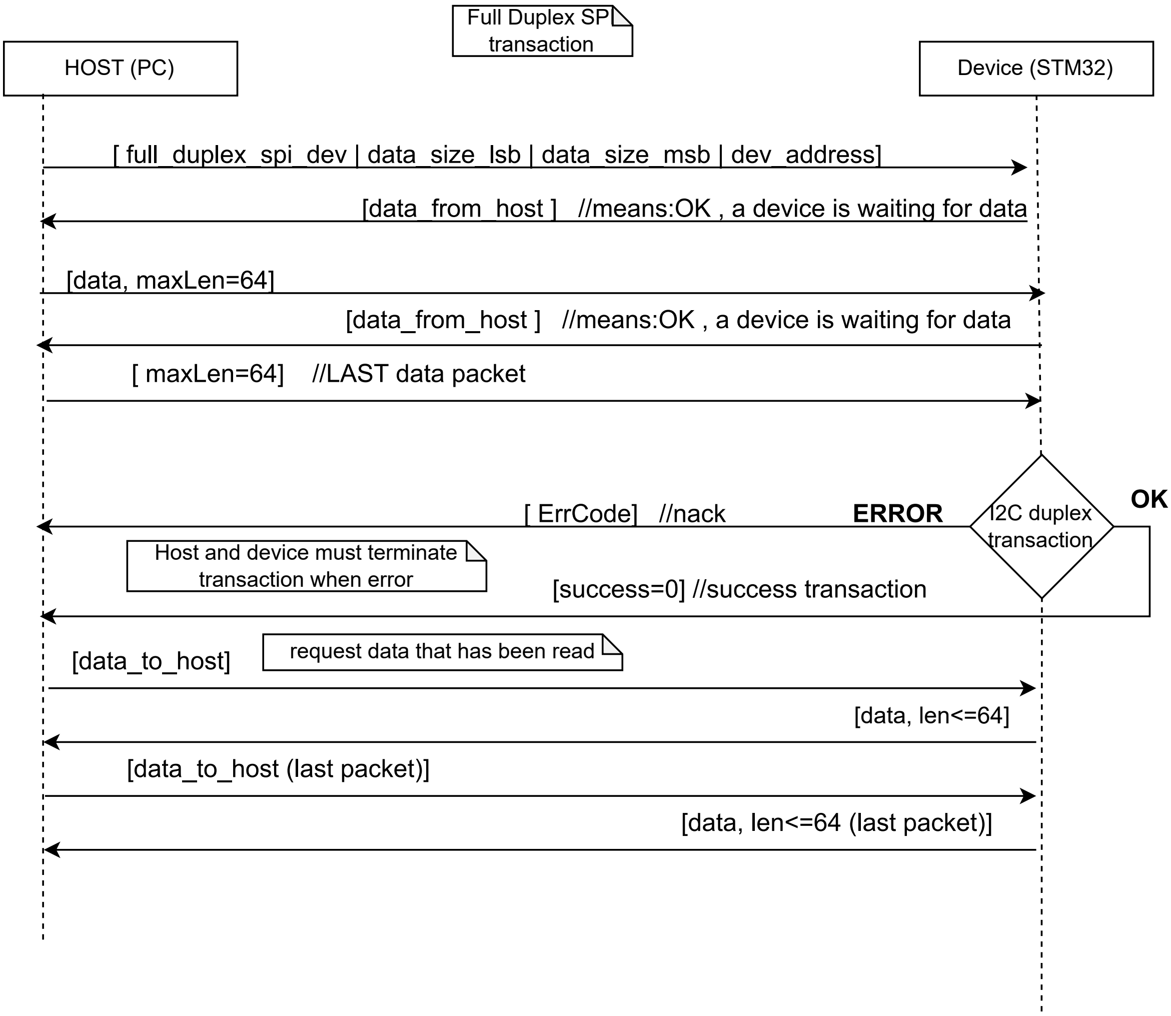

setup_interface_i2c
HOST (PC)
Device (STM32)
:[setup_interface_i2c |0|0|0| spd_b0 | spd_b1 |spd_b2|spd_b3|slave_address]
Host and device must terminate transaction when error
OK (data=0x00)

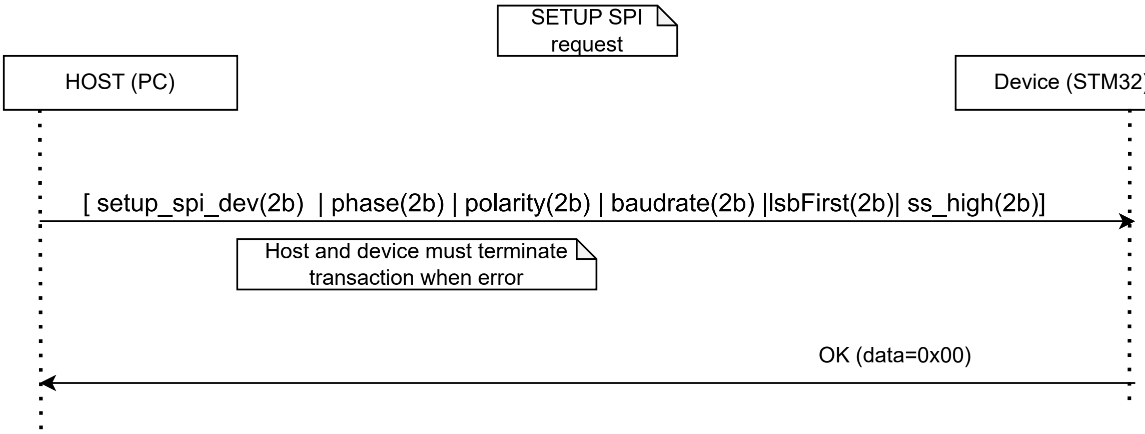

READ last received internal data from I2C1 (slave)
The host fill new data inside the internal slave I2C1 device
the "status" may be sucess or fail
HOST (PC)
Device (STM32)
[read_last_stub_rx_i2c]
[status| data_size_lsb | data_size_msb |]
[data_to_host]
[data, maxLen=64]
[data_to_host]
[data, last packet maxLen=64]


WRITE transmitter slave buffer (I2C1)
The host reads last received data from the slave I2C1 device
HOST (PC)
Device (STM32)
[write_tx_stub_buffer_i2c | data_size_lsb | data_size_msb]
[data_from_host **OR** error (0)]
[data 64bytes]
[data_to_host]
[data, maxLen=64)]


WRITE SPI request
HOST (PC)
Device (STM32)
[write_to_spi_dev | data_size_lsb | data_size_msb | dev_address]
[data_from_host] //means:OK , a device is waiting for data
[data, maxLen=64]
[data_from_host] //means:OK , a device is waiting for data
[maxLen=64] //LAST data packet
[ErrCode] //nack
ERROR
SPI transaction
OK
Host and device must terminate transaction when error
[success=0] //success transaction


READ SPI request
HOST (PC)
Device (STM32)
[read_from_spi_dev | data_size_lsb | data_size_msb | dev_address]
Host and device must terminate transaction when error
[ErrCode]
ERROR
Start read SPI transaction
OK
[success=0]
[data_to_host] ///a host is ready for new data reception
[data, len<=64]
[data_to_host (last packet)]
[data, len<=64 (last packet)]


Full Duplex SPI transaction
HOST (PC)
Device (STM32)
[full_duplex_spi_dev | data_size_lsb | data_size_msb | dev_address]
[data_from_host] //means:OK , a device is waiting for data
[data, maxLen=64]
[data_from_host] //means:OK , a device is waiting for data
[maxLen=64] //LAST data packet
[ErrCode] //nack
ERROR
I2C duplex transaction
OK
Host and device must terminate transaction when error
[success=0] //success transaction
[data_to_host]
request data that has been read
[data, len<=64]
[data_to_host (last packet)]
[data, len<=64 (last packet)]



SETUP SPI request

HOST (PC)

Device (STM32)

[setup_spi_dev(2b) | phase(2b) | polarity(2b) | baudrate(2b) |lsbFirst(2b)| ss_high(2b)]

Host and device must terminate transaction when error

OK (data=0x00)

Error codes

a)adapter_success	=0;
b)adapter_AF	=1;
c)adapter_BERR	=2;
d)adapter_ARLO	=3;
e)adapter_OVR	=4;
f)adapter_timeout	=5;
g)adapter_other_error	=6;
e)adapter_busy	=7;

Data markers

a)data_from_host	=18;
b)data_to_host	=19;

Commands

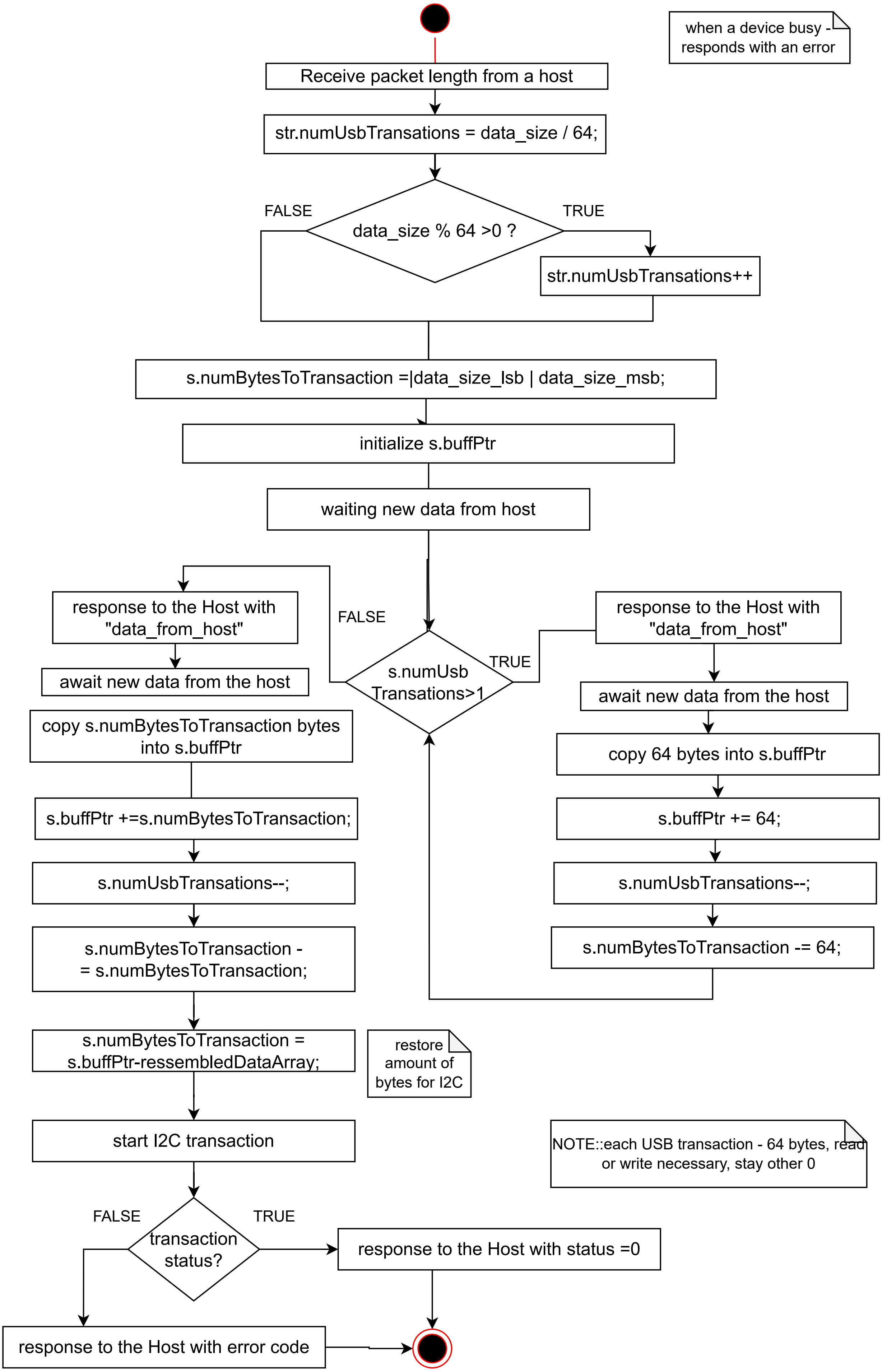
a) write_to_i2c_dev	=24;
b) read_from_i2c_dev	=25;
c) reset_interface_i2c	=26;
d) setup_interface_i2c	=27;
e) read_last_stub_rx_i2c	=28;
f) write_tx_stub_buffer_i2c	= 29;
g) write_to_spi_dev	=30
h) read_from_spi_dev	=31
i) full_duplex_spi_dev	=32
j) setup_spi_dev	=33

USB Device : the state maschine

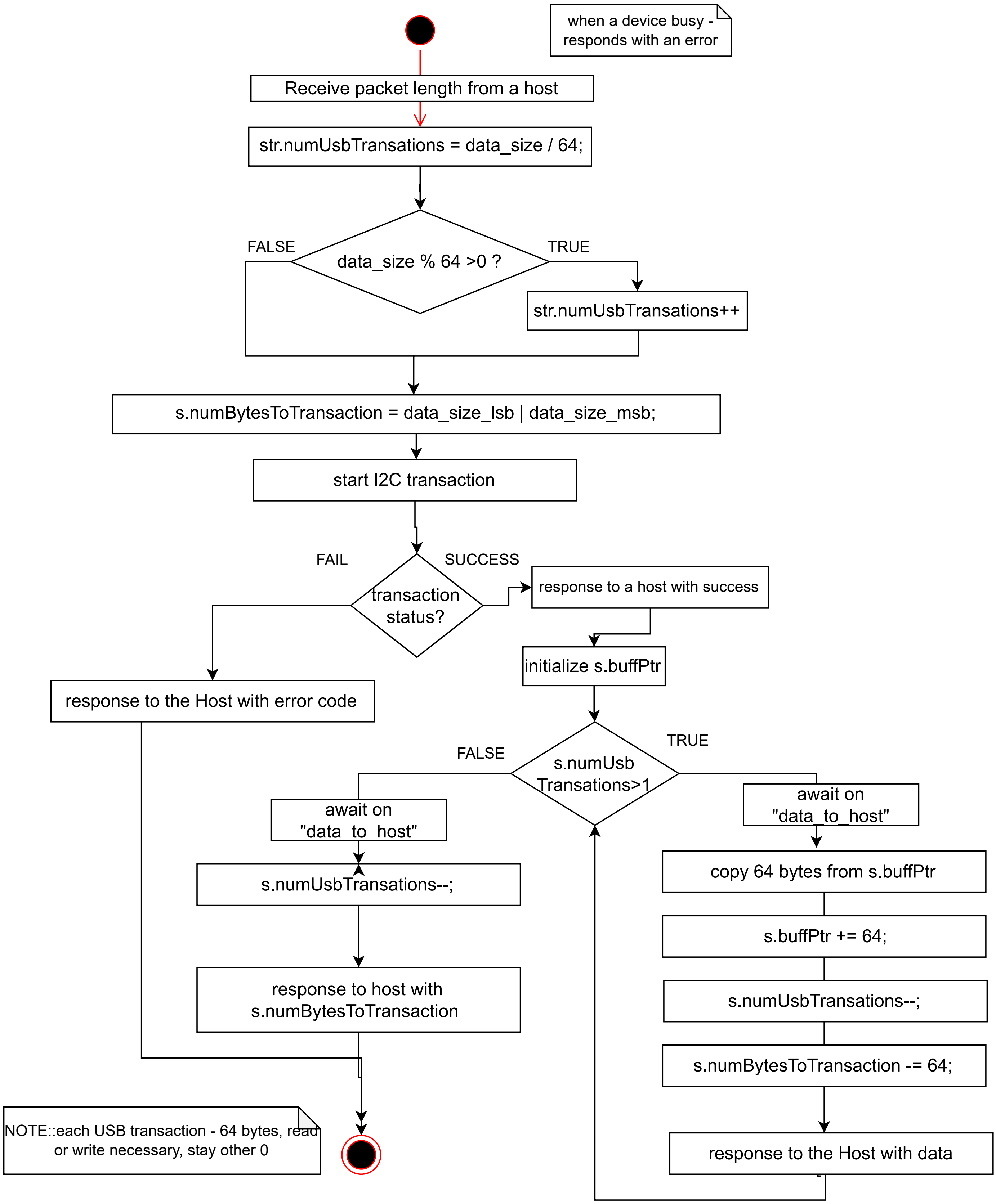
the structure must be aligned to 4 to improve CPU (Corerx-M3) and peripheral performance

```
STRUCT {
uchar typeOfAction;
uchar slaveAddr;
uchar* buffPtr; //changes during exec.
uin16_t numUsbTransations;
uint16_t numBytesToTransaction;
uchar*  reassembledDataArray; //not changed
}statesHandle;
```

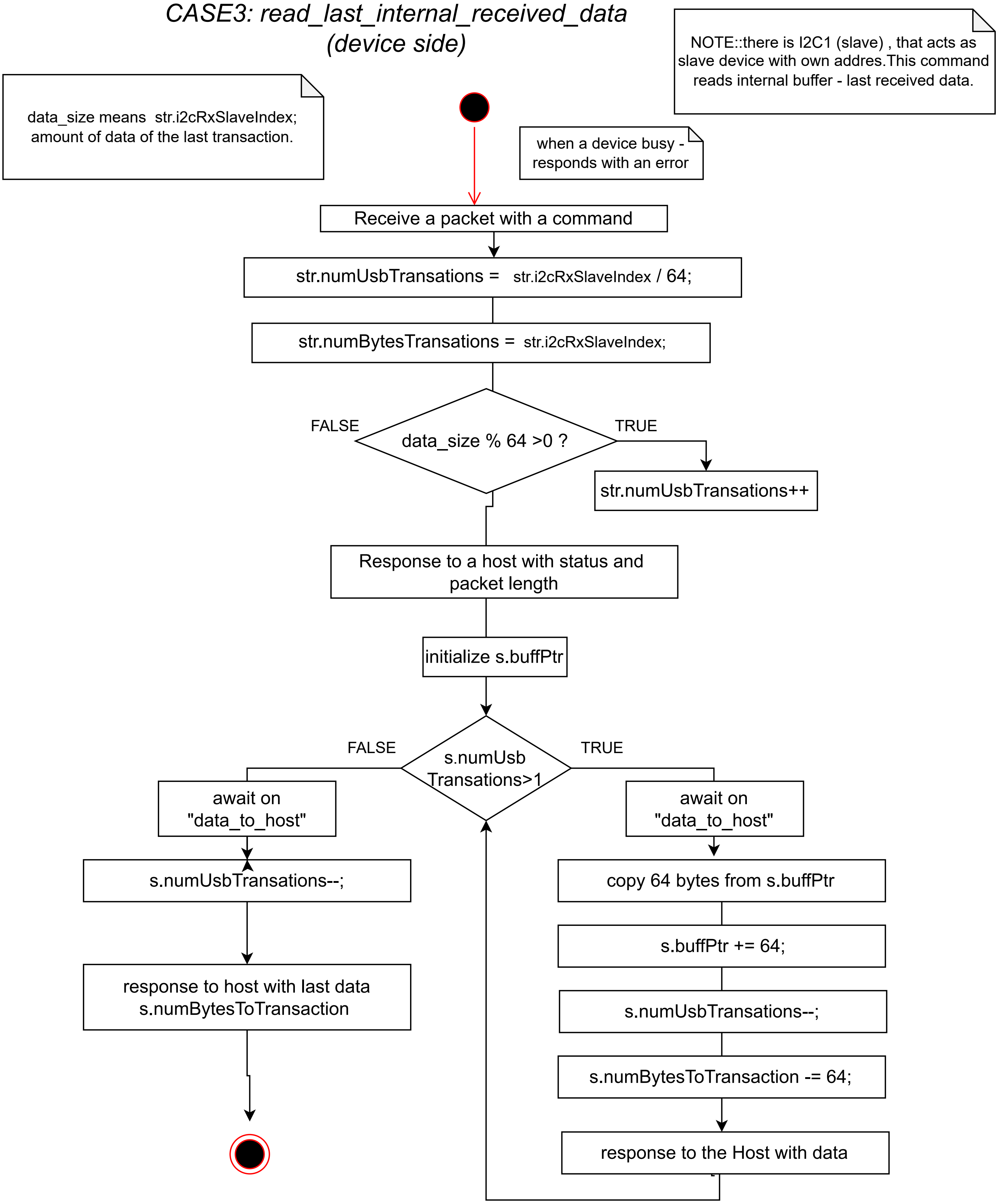
CASE1: write_to_i2c_dev (device side)


when a device busy - responds with an error
Receive packet length from a host
str.numUsbTransations = data_size / 64;
FALSE
data_size % 64 >0 ?
TRUE
str.numUsbTransations++
s.numBytesToTransaction =|data_size_lsb | data_size_msb;
initialize s.buffPtr
waiting new data from host
response to the Host with "data_from_host"
await new data from the host
copy s.numBytesToTransaction bytes into s.buffPtr
s.buffPtr +=s.numBytesToTransaction;
s.numUsbTransations--;
s.numBytesToTransaction -= s.numBytesToTransaction;
s.numBytesToTransaction = s.buffPtr-ressembledDataArray;
FALSE
s.numUsb Transations>1
TRUE
response to the Host with "data_from_host"
await new data from the host
copy 64 bytes into s.buffPtr
s.buffPtr += 64;
s.numUsbTransations--;
s.numBytesToTransaction -= 64;
restore amount of bytes for I2C
start I2C transaction
NOTE::each USB transaction - 64 bytes, read or write necessary, stay other 0
FALSE
TRUE
transaction status?
response to the Host with status =0
response to the Host with error code

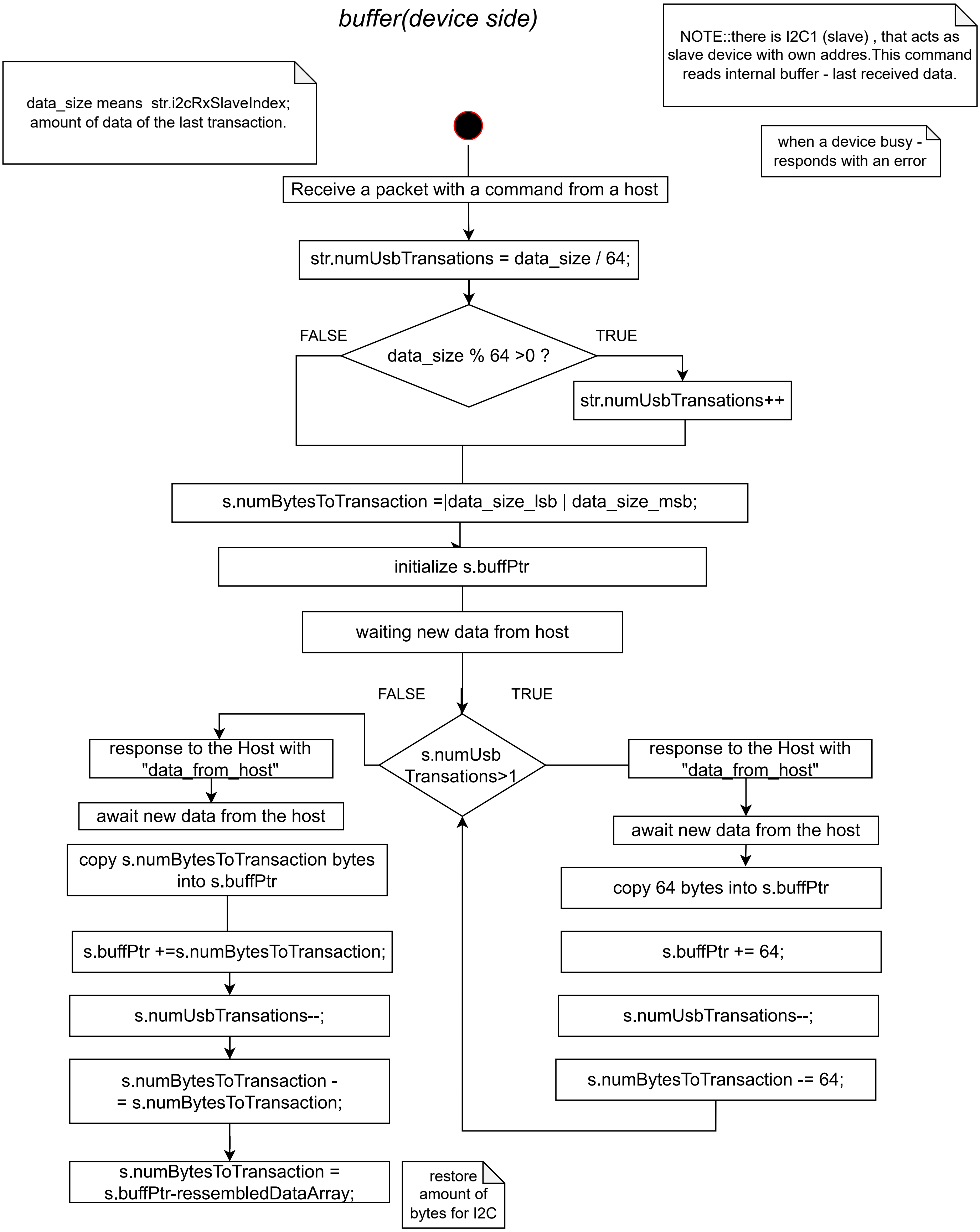
CASE2: read_from_i2c_dev (device side)


when a device busy - responds with an error
Receive packet length from a host
str.numUsbTransations = data_size / 64;
FALSE
data_size % 64 >0 ?
TRUE
str.numUsbTransations++
s.numBytesToTransaction = data_size_lsb | data_size_msb;
start I2C transaction
FAIL
SUCCESS
transaction status?
response to a host with success
initialize s.buffPtr
response to the Host with error code
FALSE
s.numUsb Transations>1
TRUE
await on "data_to_host"
await on "data_to_host"
s.numUsbTransations--;
copy 64 bytes from s.buffPtr
s.buffPtr += 64;
response to host with s.numBytesToTransaction
s.numUsbTransations--;
s.numBytesToTransaction -= 64;
NOTE::each USB transaction - 64 bytes, read or write necessary, stay other 0
response to the Host with data

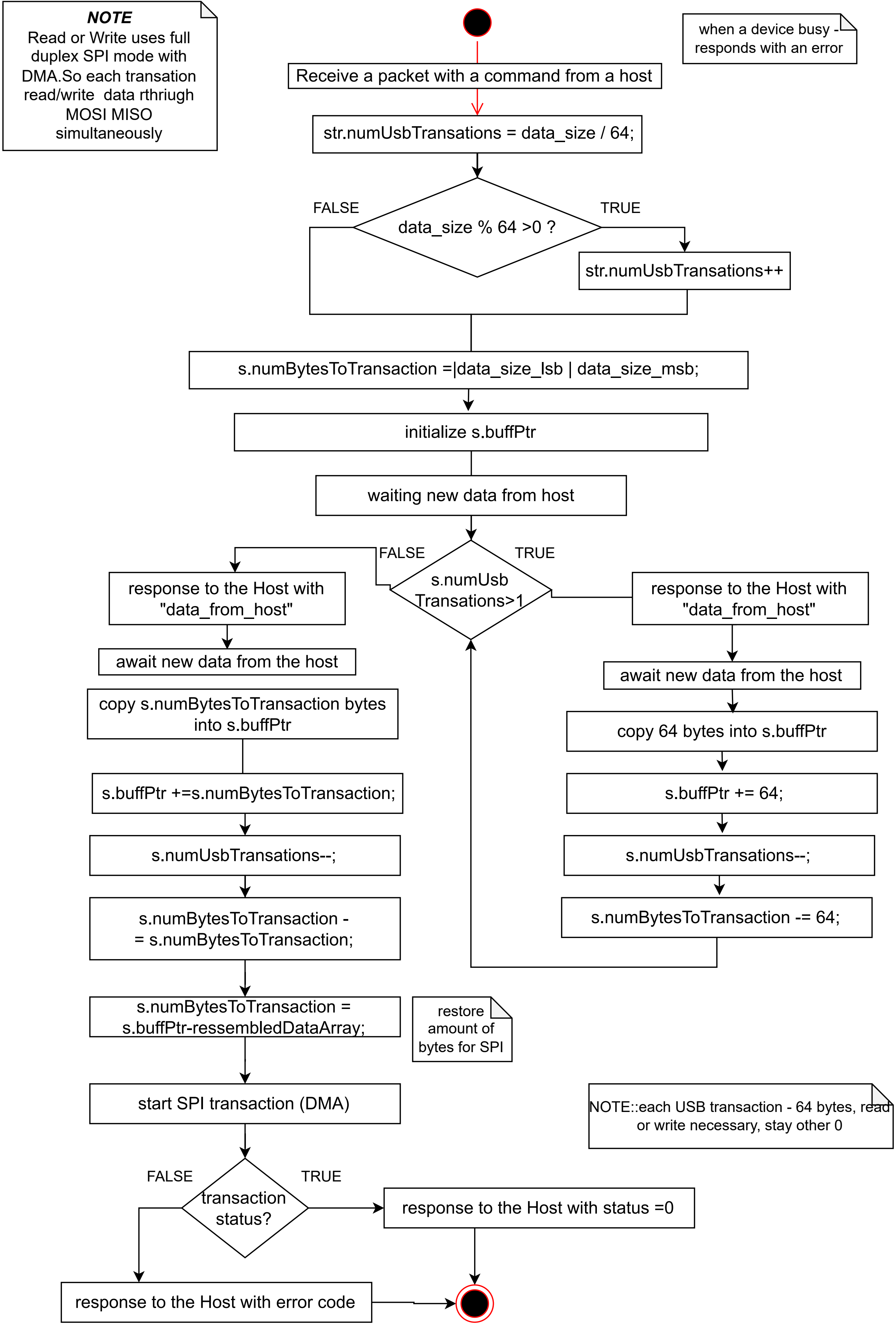
CASE3: read_last_internal_received_data (device side)


NOTE::there is I2C1 (slave) , that acts as slave device with own addres.This command reads internal buffer - last received data.
data_size means str.i2cRxSlaveIndex; amount of data of the last transaction.
when a device busy - responds with an error
Receive a packet with a command
str.numUsbTransations = str.i2cRxSlaveIndex / 64;
str.numBytesTransations = str.i2cRxSlaveIndex;
FALSE
data_size % 64 >0 ?
TRUE
str.numUsbTransations++
Response to a host with status and packet length
initialize s.buffPtr
FALSE
s.numUsb Transations>1
TRUE
await on "data_to_host"
s.numUsbTransations--;
response to host with last data s.numBytesToTransaction
await on "data_to_host"
copy 64 bytes from s.buffPtr
s.buffPtr += 64;
s.numUsbTransations--;
s.numBytesToTransaction -= 64;
response to the Host with data

CASE4: Write slave Tx internal buffer(device side)


NOTE::there is I2C1 (slave) , that acts as slave device with own addres.This command reads internal buffer - last received data.
data_size means str.i2cRxSlaveIndex; amount of data of the last transaction.
when a device busy - responds with an error
Receive a packet with a command from a host
str.numUsbTransations = data_size / 64;
FALSE
data_size % 64 >0 ?
TRUE
str.numUsbTransations++
s.numBytesToTransaction =|data_size_lsb | data_size_msb;
initialize s.buffPtr
waiting new data from host
FALSE
TRUE
s.numUsb Transations>1
response to the Host with "data_from_host"
await new data from the host
copy s.numBytesToTransaction bytes into s.buffPtr
s.buffPtr +=s.numBytesToTransaction;
s.numUsbTransations--;
s.numBytesToTransaction -= s.numBytesToTransaction;
s.numBytesToTransaction = s.buffPtr-ressembledDataArray;
restore amount of bytes for I2C
response to the Host with "data_from_host"
await new data from the host
copy 64 bytes into s.buffPtr
s.buffPtr += 64;
s.numUsbTransations--;
s.numBytesToTransaction -= 64;

CASE5: write_to_spi_dev *(device side)*


Receive a packet with a command from a host
str.numUsbTransations = data_size / 64;
data_size % 64 >0 ?
FALSE
TRUE
str.numUsbTransations++
s.numBytesToTransaction =|data_size_lsb | data_size_msb;
initialize s.buffPtr
waiting new data from host
s.numUsb Transations>1
FALSE
TRUE
response to the Host with "data_from_host"
await new data from the host
copy s.numBytesToTransaction bytes into s.buffPtr
s.buffPtr +=s.numBytesToTransaction;
s.numUsbTransations--;
s.numBytesToTransaction -= s.numBytesToTransaction;
s.numBytesToTransaction = s.buffPtr-ressembledDataArray;
start SPI transaction (DMA)
response to the Host with "data_from_host"
await new data from the host
copy 64 bytes into s.buffPtr
s.buffPtr += 64;
s.numUsbTransations--;
s.numBytesToTransaction -= 64;
transaction status?
FALSE
TRUE
response to the Host with status =0
response to the Host with error code

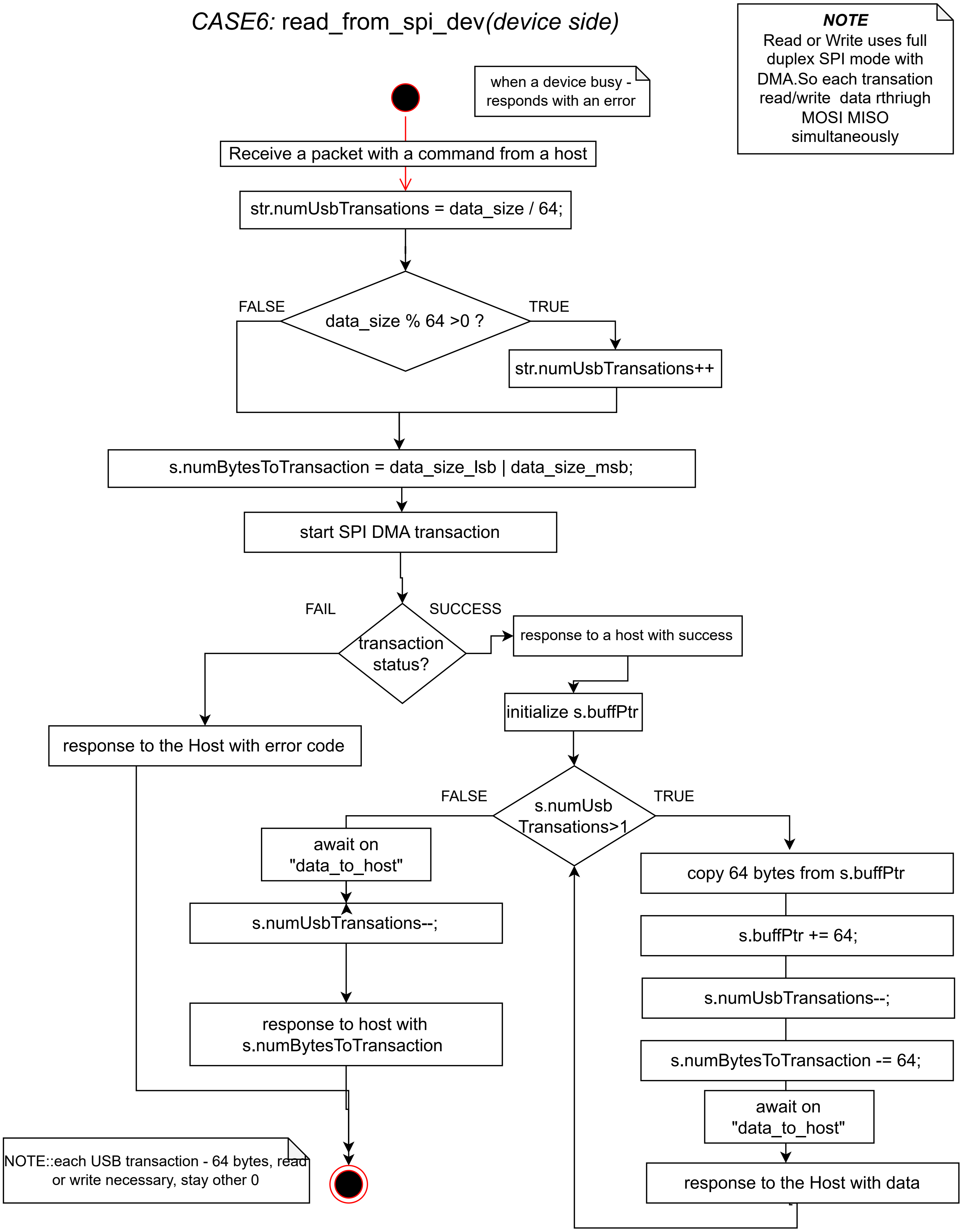
NOTE
Read or Write uses full duplex SPI mode with DMA.So each transation read/write data rthriugh MOSI MISO simultaneously

when a device busy - responds with an error

restore amount of bytes for SPI

NOTE::each USB transaction - 64 bytes, read or write necessary, stay other 0

CASE6: read_from_spi_dev*(device side)*

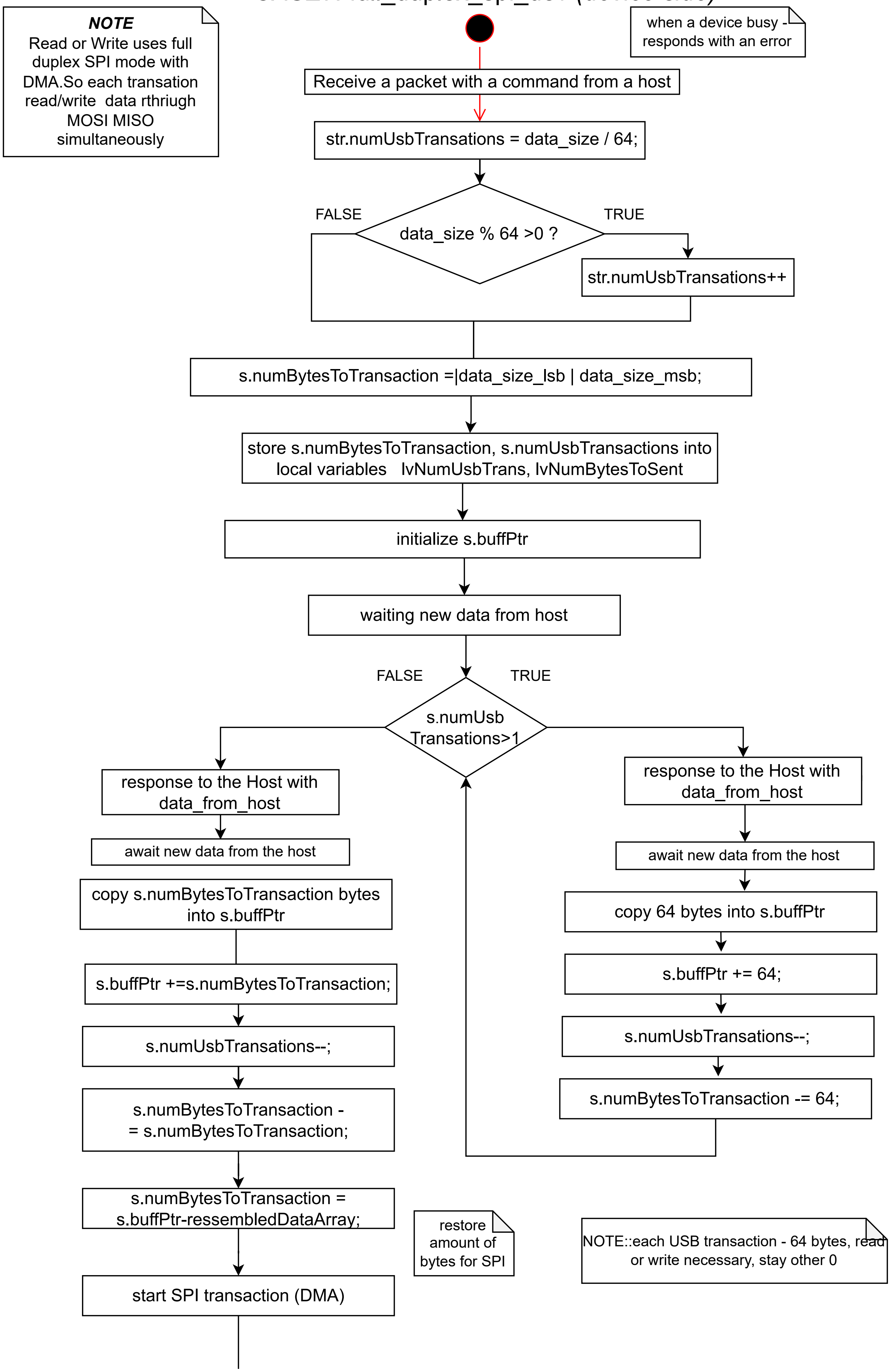

Receive a packet with a command from a host
str.numUsbTransations = data_size / 64;
data_size % 64 >0 ?
FALSE
TRUE
str.numUsbTransations++
s.numBytesToTransaction = data_size_lsb | data_size_msb;
start SPI DMA transaction
transaction status?
FAIL
SUCCESS
response to a host with success
initialize s.buffPtr
response to the Host with error code
s.numUsb Transations>1
FALSE
TRUE
await on "data_to_host"
s.numUsbTransations--;
response to host with s.numBytesToTransaction
copy 64 bytes from s.buffPtr
s.buffPtr += 64;
s.numUsbTransations--;
s.numBytesToTransaction -= 64;
await on "data_to_host"
response to the Host with data

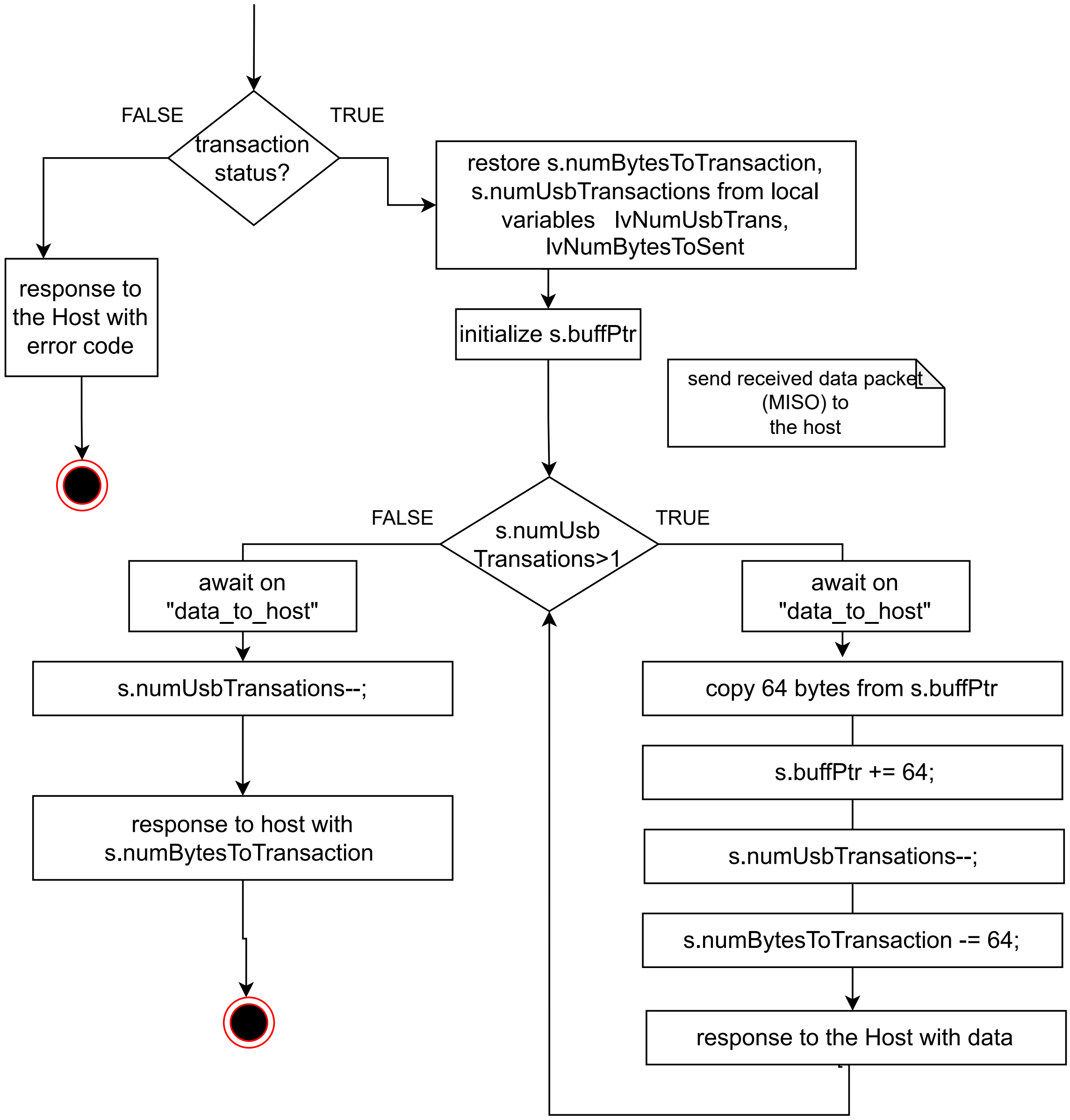
when a device busy - responds with an error

NOTE
Read or Write uses full duplex SPI mode with DMA.So each transation read/write data rthriugh MOSI MISO simultaneously

NOTE::each USB transaction - 64 bytes, read or write necessary, stay other 0

CASE7: full_duplex_spi_dev *(device side)*


NOTE
Read or Write uses full duplex SPI mode with DMA.So each transation read/write data rthriugh MOSI MISO simultaneously
when a device busy - responds with an error
Receive a packet with a command from a host
str.numUsbTransations = data_size / 64;
FALSE
data_size % 64 >0 ?
TRUE
str.numUsbTransations++
s.numBytesToTransaction =|data_size_lsb | data_size_msb;
store s.numBytesToTransaction, s.numUsbTransactions into local variables lvNumUsbTrans, lvNumBytesToSent
initialize s.buffPtr
waiting new data from host
FALSE
TRUE
s.numUsb Transations>1
response to the Host with data_from_host
await new data from the host
copy s.numBytesToTransaction bytes into s.buffPtr
s.buffPtr +=s.numBytesToTransaction;
s.numUsbTransations--;
s.numBytesToTransaction -= s.numBytesToTransaction;
s.numBytesToTransaction = s.buffPtr-ressembledDataArray;
start SPI transaction (DMA)
restore amount of bytes for SPI
response to the Host with data_from_host
await new data from the host
copy 64 bytes into s.buffPtr
s.buffPtr += 64;
s.numUsbTransations--;
s.numBytesToTransaction -= 64;
NOTE::each USB transaction - 64 bytes, read or write necessary, stay other 0


FALSE
transaction status?
TRUE
restore s.numBytesToTransaction, s.numUsbTransactions from local variables lvNumUsbTrans, lvNumBytesToSent
response to the Host with error code
initialize s.buffPtr
send received data packet (MISO) to the host
FALSE
s.numUsb Transations>1
TRUE
await on "data_to_host"
await on "data_to_host"
s.numUsbTransations--;
copy 64 bytes from s.buffPtr
s.buffPtr += 64;
response to host with s.numBytesToTransaction
s.numUsbTransations--;
s.numBytesToTransaction -= 64;
response to the Host with data

USB Device : the state maschine

STRUCT {
uchar typeOfAction;
uchar slaveAddr;
uchar* buffPtr; *//changes during exec.*
uin16_t numUsbTransations;
uint16_t numBytesToTransaction;
uchar* reassembledDataArray; *//not changed*
}statesHandle;

Error codes

a)Success	=0;
b)Acknowledge failure AF	=1;
c)Bus Error BERR	=2;
d)Arbitration lost (ARLO)	=3;
e)Overrun/underrun (OVR)	=4;
f)Timeout	=5;
g)Other error	=6;
e)Busy	=7;

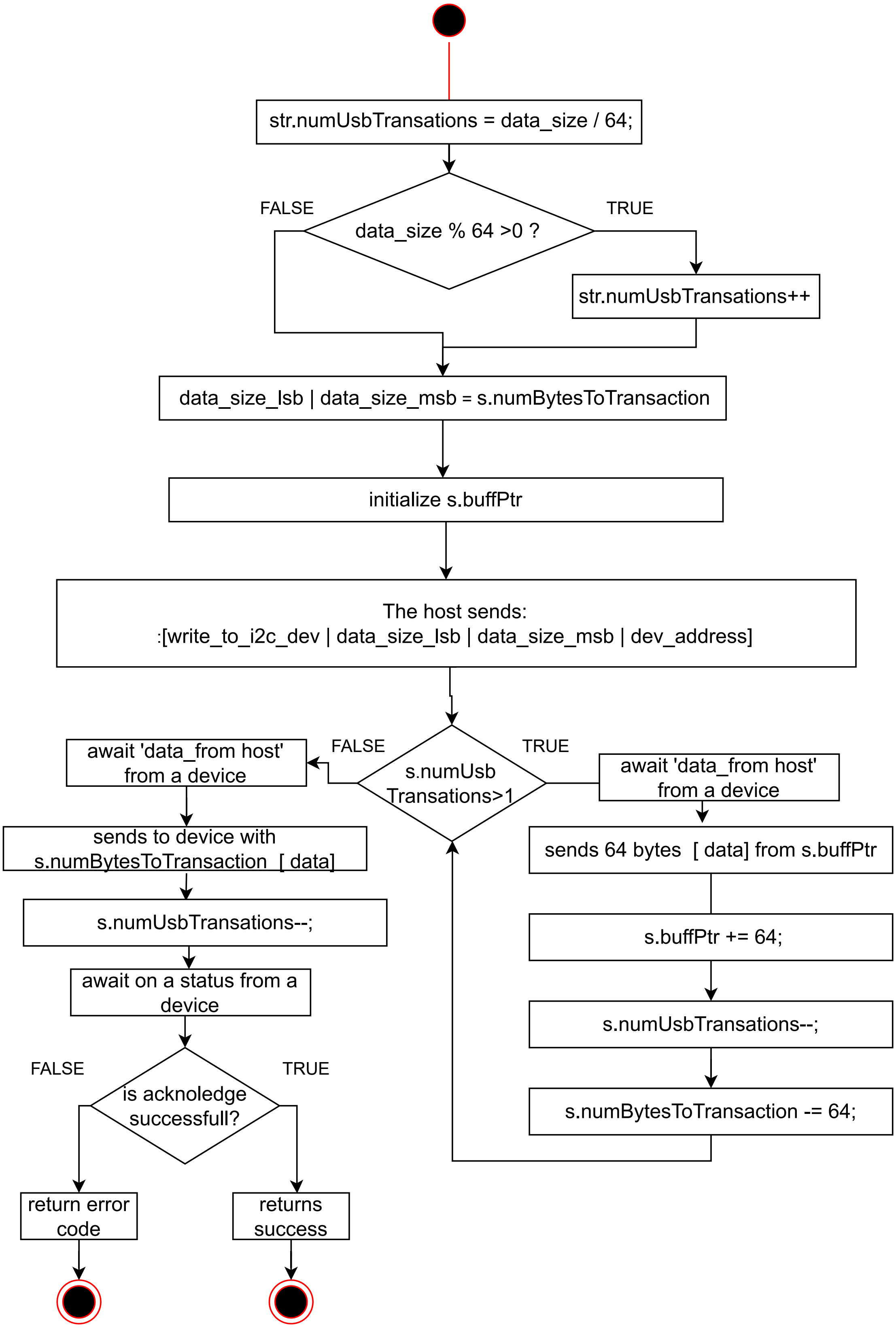
Data markers

a)data_from_host	=18;
b)data_to_host	=19;

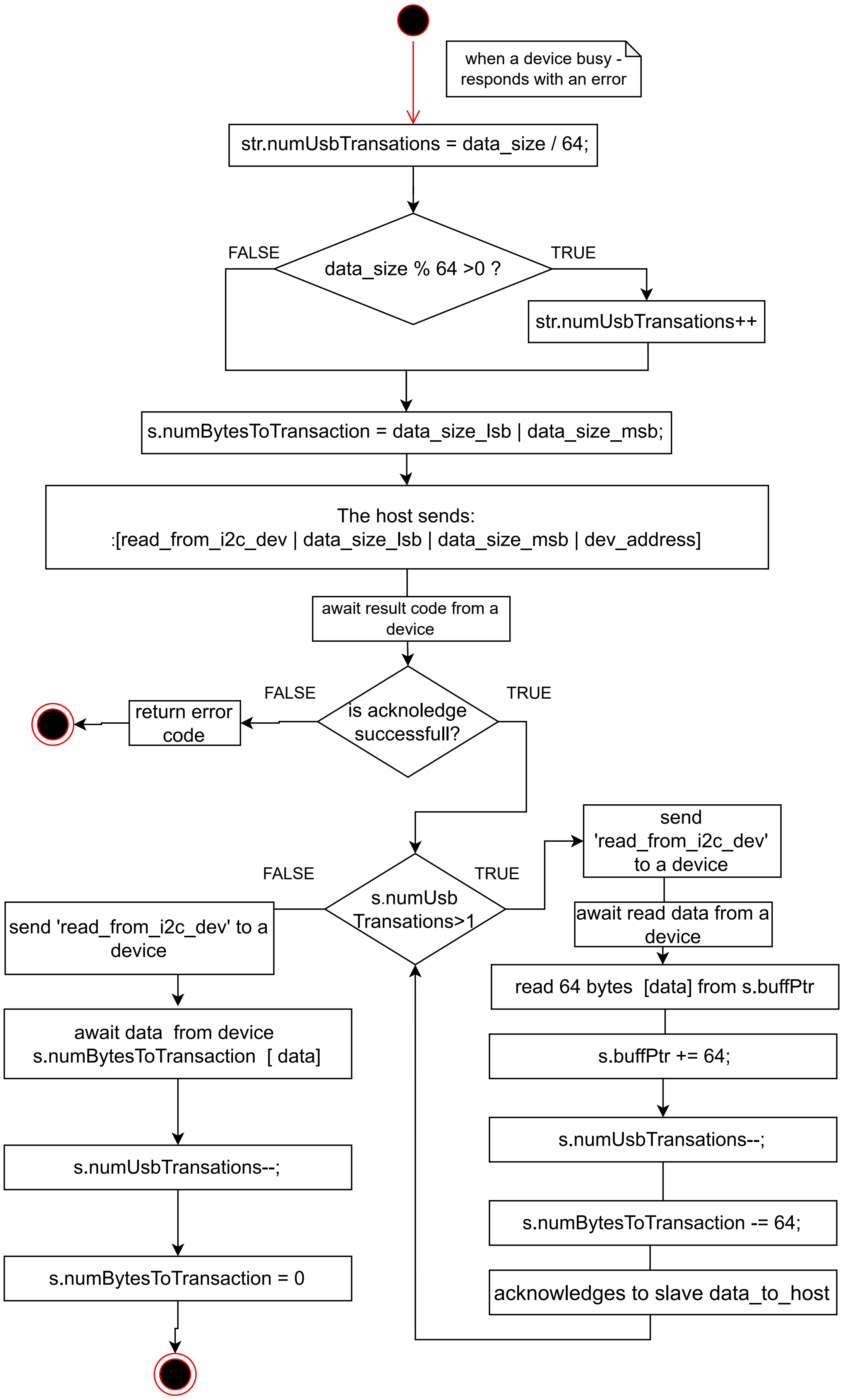
Commands

a) write_to_i2c_dev	=24;
b) read_from_i2c_dev	=25;
c) reset_interface_i2c	=26;
d) setup_interface_i2c	=27;
e) read_last_stub_rx_i2c	=28;
f) write_tx_stub_buffer_i2c	= 29;
g) write_to_spi_dev	=30
h) read_from_spi_dev	=31
i) full_duplex_spi_dev	=32
j) setup_spi_dev	=33

CASE1: write_to_i2c_dev (HOST side)

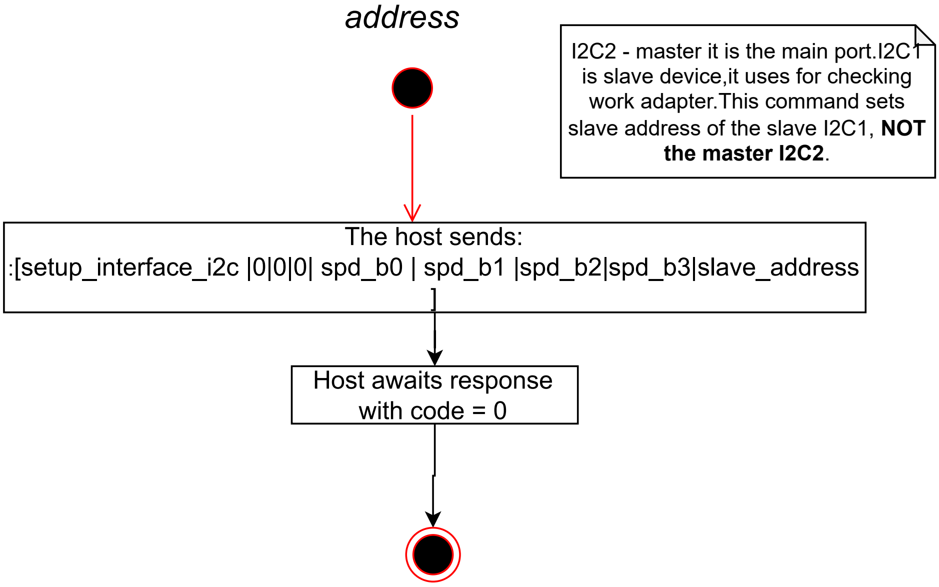

str.numUsbTransations = data_size / 64;
data_size % 64 >0 ?
FALSE
TRUE
str.numUsbTransations++
data_size_lsb | data_size_msb = s.numBytesToTransaction
initialize s.buffPtr
The host sends:
:[write_to_i2c_dev | data_size_lsb | data_size_msb | dev_address]
s.numUsb Transations>1
FALSE
TRUE
await 'data_from host' from a device
sends to device with s.numBytesToTransaction [data]
s.numUsbTransations--;
await on a status from a device
await 'data_from host' from a device
sends 64 bytes [data] from s.buffPtr
s.buffPtr += 64;
s.numUsbTransations--;
s.numBytesToTransaction -= 64;
is acknoledge successfull?
FALSE
TRUE
return error code
returns success

CASE2: read_from_i2c_dev (HOST side)



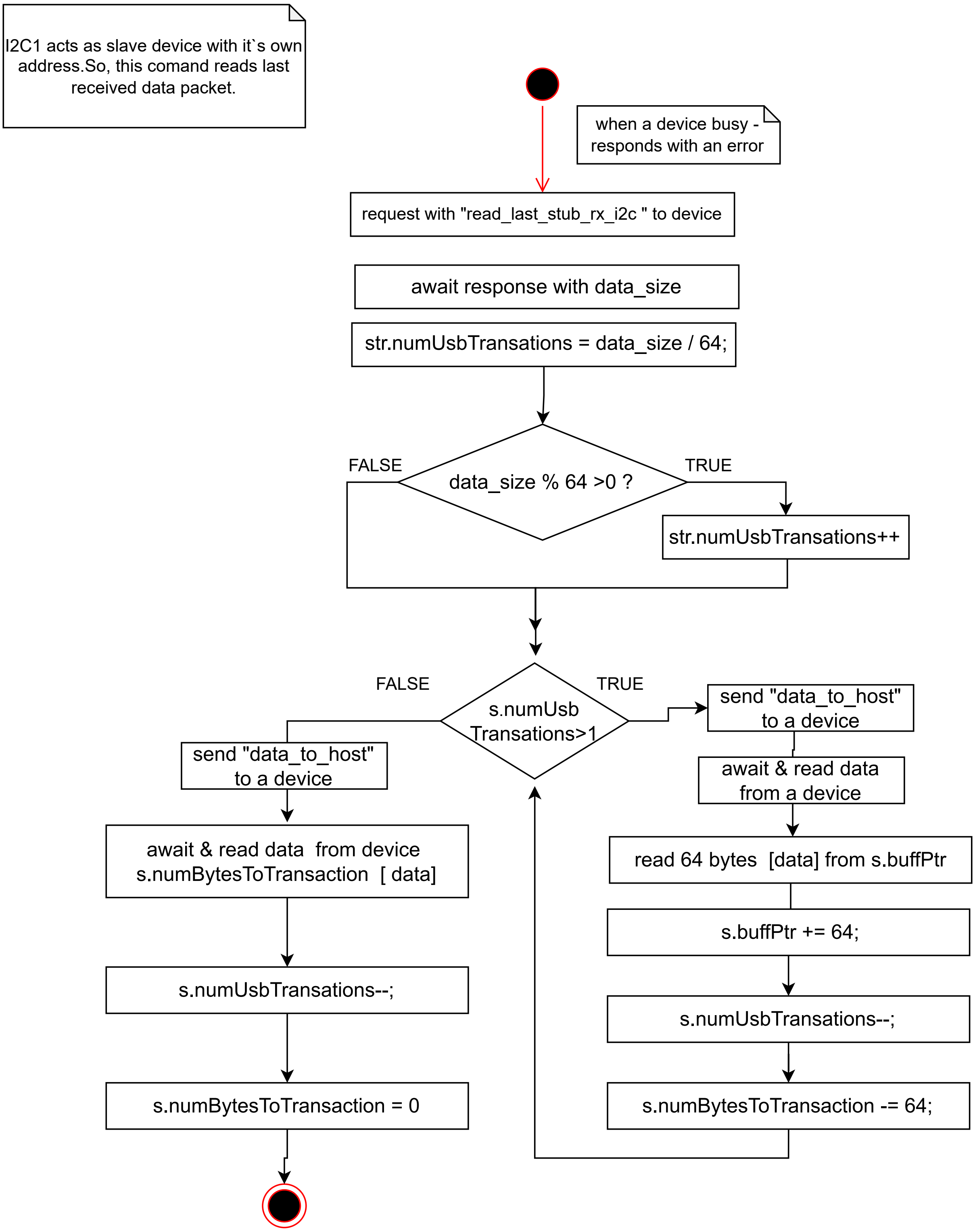
when a device busy - responds with an error
str.numUsbTransations = data_size / 64;
data_size % 64 >0 ?
FALSE
TRUE
str.numUsbTransations++
s.numBytesToTransaction = data_size_lsb | data_size_msb;
The host sends:
:[read_from_i2c_dev | data_size_lsb | data_size_msb | dev_address]
await result code from a device
is acknoledge successfull?
FALSE
TRUE
return error code
s.numUsb Transations>1
FALSE
TRUE
send 'read_from_i2c_dev' to a device
send 'read_from_i2c_dev' to a device
await read data from a device
read 64 bytes [data] from s.buffPtr
s.buffPtr += 64;
s.numUsbTransations--;
s.numBytesToTransaction -= 64;
acknowledges to slave data_to_host
await data from device s.numBytesToTransaction [data]
s.numUsbTransations--;
s.numBytesToTransaction = 0

Set frequency and I2C1 slave address


The host sends:
:[setup_interface_i2c |0|0|0| spd_b0 | spd_b1 |spd_b2|spd_b3|slave_address]
Host awaits response with code = 0

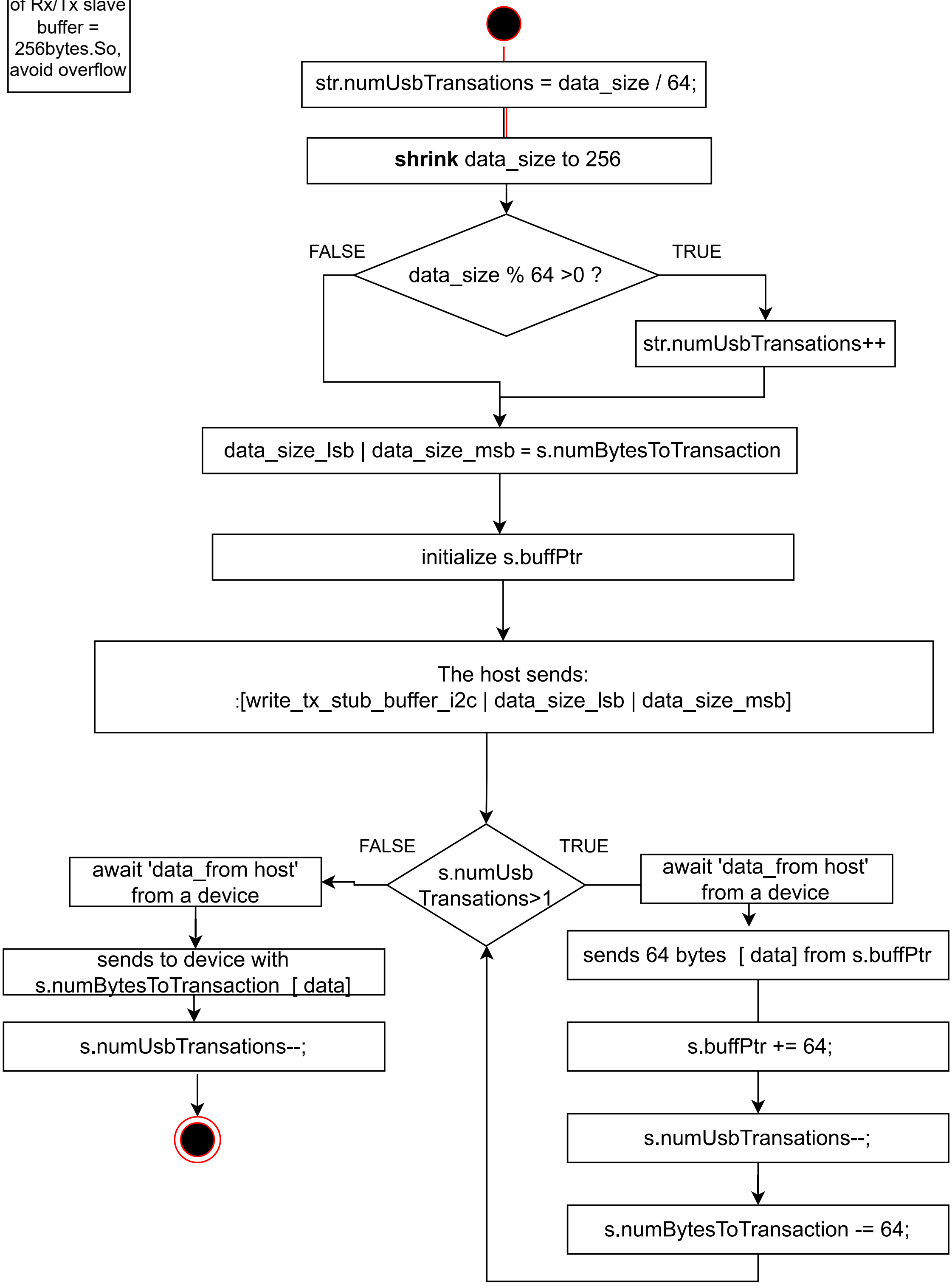
I2C2 - master it is the main port.I2C1 is slave device,it uses for checking work adapter.This command sets slave address of the slave I2C1, **NOT the master I2C2**.

CASE3: read_last_received_internal_data (HOST side)

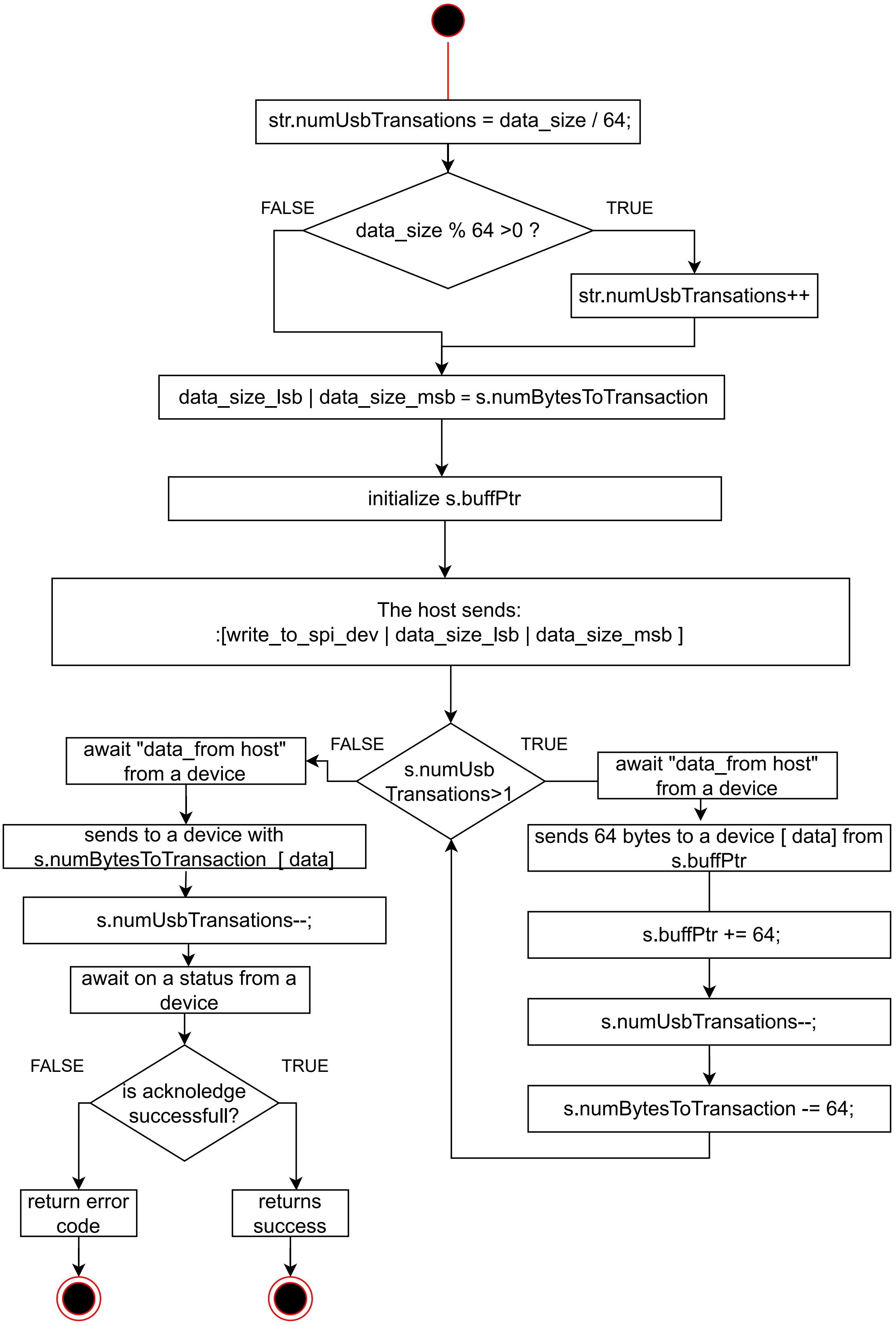

I2C1 acts as slave device with it`s own address.So, this comand reads last received data packet.
when a device busy - responds with an error
request with "read_last_stub_rx_i2c " to device
await response with data_size
str.numUsbTransations = data_size / 64;
FALSE
data_size % 64 >0 ?
TRUE
str.numUsbTransations++
FALSE
s.numUsb Transations>1
TRUE
send "data_to_host" to a device
await & read data from a device
read 64 bytes [data] from s.buffPtr
s.buffPtr += 64;
s.numUsbTransations--;
s.numBytesToTransaction -= 64;
send "data_to_host" to a device
await & read data from device s.numBytesToTransaction [data]
s.numUsbTransations--;
s.numBytesToTransaction = 0

NOTE: maximum size of Rx/Tx slave buffer = 256bytes.So, avoid overflow

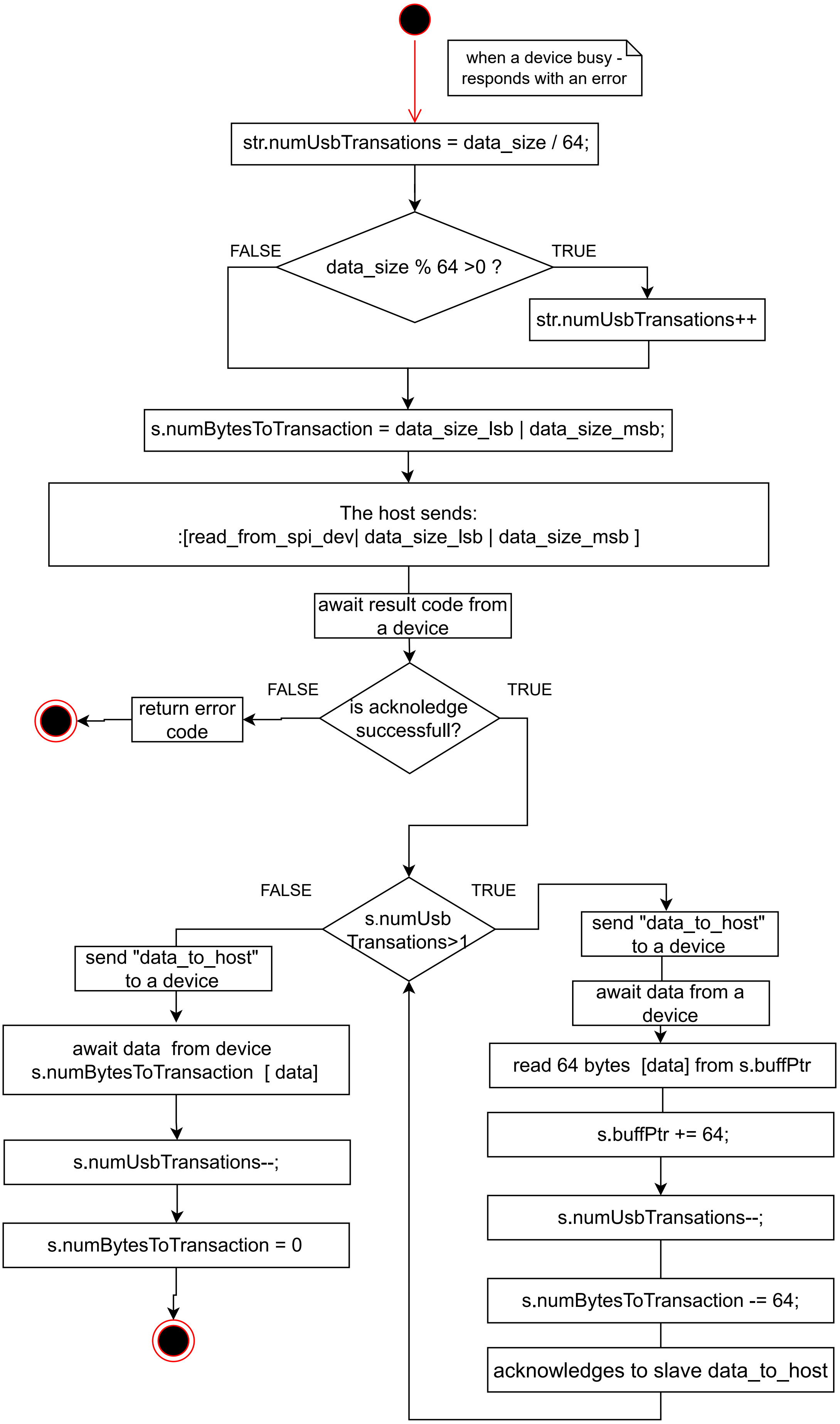
CASE4: write internal slave Tx bufer (I2C1) (HOST side)


str.numUsbTransations = data_size / 64;
shrink data_size to 256
data_size % 64 >0 ?
FALSE
TRUE
str.numUsbTransations++
data_size_lsb | data_size_msb = s.numBytesToTransaction
initialize s.buffPtr
The host sends:
:[write_tx_stub_buffer_i2c | data_size_lsb | data_size_msb]
s.numUsb Transations>1
FALSE
TRUE
await 'data_from host' from a device
sends to device with s.numBytesToTransaction [data]
s.numUsbTransations--;
await 'data_from host' from a device
sends 64 bytes [data] from s.buffPtr
s.buffPtr += 64;
s.numUsbTransations--;
s.numBytesToTransaction -= 64;

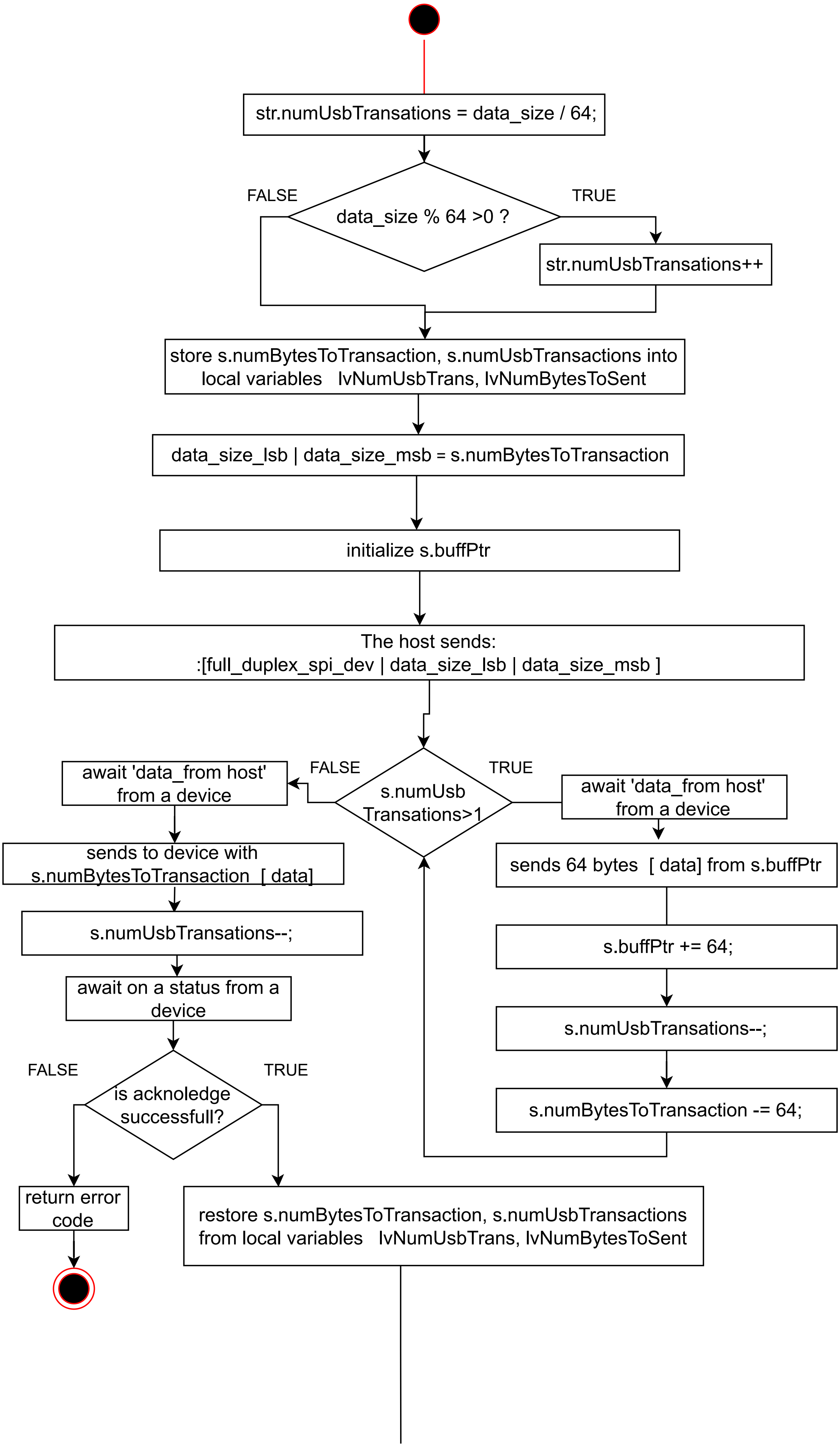
CASE5: write_to_spi_dev *(HOST side)*

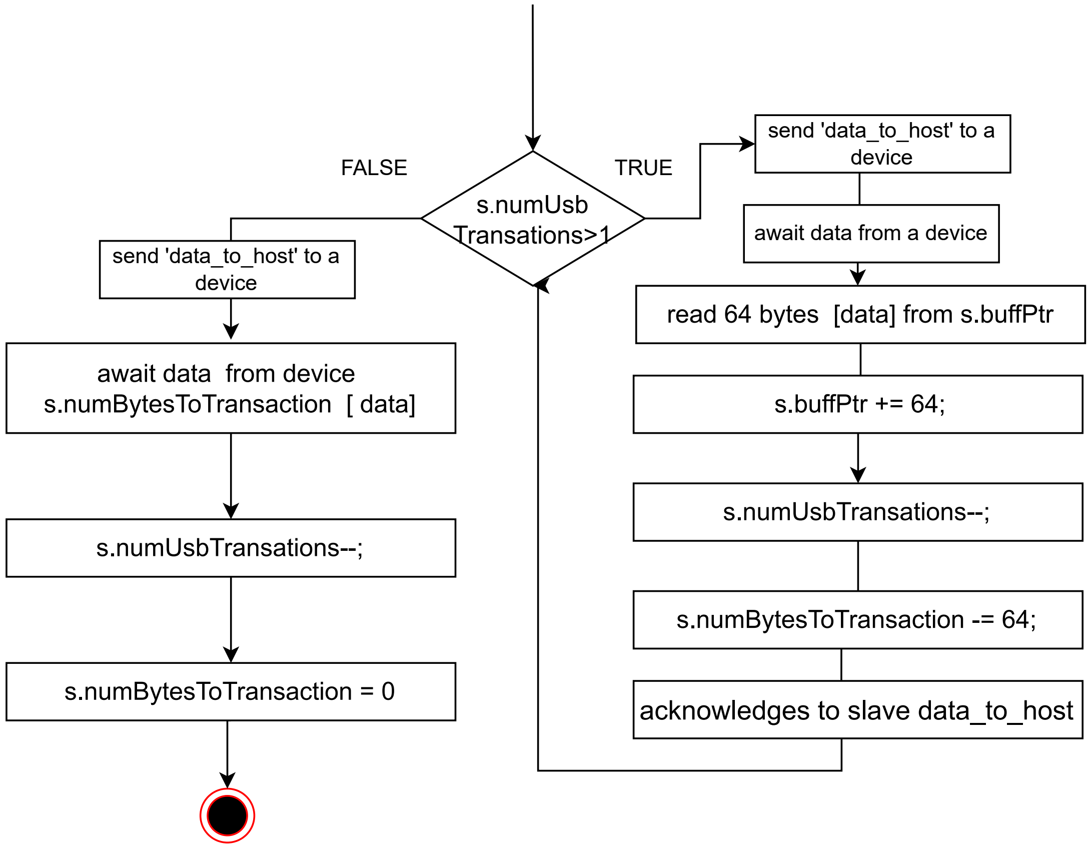

str.numUsbTransations = data_size / 64;
data_size % 64 >0 ?
FALSE
TRUE
str.numUsbTransations++
data_size_lsb | data_size_msb = s.numBytesToTransaction
initialize s.buffPtr
The host sends:
:[write_to_spi_dev | data_size_lsb | data_size_msb]
s.numUsb Transations>1
FALSE
TRUE
await "data_from host" from a device
sends to a device with s.numBytesToTransaction [data]
s.numUsbTransations--;
await on a status from a device
is acknoledge successfull?
FALSE
TRUE
return error code
returns success
await "data_from host" from a device
sends 64 bytes to a device [data] from s.buffPtr
s.buffPtr += 64;
s.numUsbTransations--;
s.numBytesToTransaction -= 64;

CASE6: read_from_spi_dev*(HOST side)*


when a device busy - responds with an error
str.numUsbTransations = data_size / 64;
data_size % 64 >0 ?
FALSE
TRUE
str.numUsbTransations++
s.numBytesToTransaction = data_size_lsb | data_size_msb;
The host sends:
:[read_from_spi_dev| data_size_lsb | data_size_msb]
await result code from a device
is acknoledge successfull?
FALSE
TRUE
return error code
s.numUsb Transations>1
FALSE
TRUE
send "data_to_host" to a device
await data from device s.numBytesToTransaction [data]
s.numUsbTransations--;
s.numBytesToTransaction = 0
send "data_to_host" to a device
await data from a device
read 64 bytes [data] from s.buffPtr
s.buffPtr += 64;
s.numUsbTransations--;
s.numBytesToTransaction -= 64;
acknowledges to slave data_to_host

CASE7: full_duplex_spi_dev *(HOST side)*


str.numUsbTransations = data_size / 64;
FALSE
data_size % 64 >0 ?
TRUE
str.numUsbTransations++
store s.numBytesToTransaction, s.numUsbTransactions into local variables lvNumUsbTrans, lvNumBytesToSent
data_size_lsb | data_size_msb = s.numBytesToTransaction
initialize s.buffPtr
The host sends:
:[full_duplex_spi_dev | data_size_lsb | data_size_msb]
await 'data_from host' from a device
FALSE
s.numUsb Transations>1
TRUE
await 'data_from host' from a device
sends to device with s.numBytesToTransaction [data]
sends 64 bytes [data] from s.buffPtr
s.numUsbTransations--;
s.buffPtr += 64;
await on a status from a device
s.numUsbTransations--;
FALSE
TRUE
is acknoledge successfull?
s.numBytesToTransaction -= 64;
return error code
restore s.numBytesToTransaction, s.numUsbTransactions from local variables lvNumUsbTrans, lvNumBytesToSent


FALSE
TRUE
s.numUsb Transations>1
send 'data_to_host' to a device
await data from device s.numBytesToTransaction [data]
s.numUsbTransations--;
s.numBytesToTransaction = 0
send 'data_to_host' to a device
await data from a device
read 64 bytes [data] from s.buffPtr
s.buffPtr += 64;
s.numUsbTransations--;
s.numBytesToTransaction -= 64;
acknowledges to slave data_to_host