***BUTTERFLY*LABS**

BitFORCE SC Communication Protocol
Revision 2.2.0 – Preliminary

WARNING:

This document is in draft stage. It will be subject to changes until preliminary and final revisions are published.

March 2013

SecureHASH:
b9a199a934c5522bc00f97b2cce74ec69dc642abe1b95e5585b4816f4c04d5cb

1. Introduction

The purpose of this document is to examine the communication protocol between a host device, such as a PC, Mac or any device capable of operating as a host and use FTDI driver to communicate with BitFORCE SC series.

It must be noted that the BitFORCE SC uses FTDI USB engine to establish connection with its host, and thus having the FTDI drivers installed on the system is mandatory. Otherwise, the host may not recognize the device.

FTDI Drivers are available at http://www.ftdichip.com/Drivers/VCP.htm

2. Backward Compatibility

The BitFORCE SC devices are generally not compatible with the older FPGA family of products. The only exception is the "Device Identification" command, as this command remains valid with the new series of BitFORCE SC. Please refer to the "Commands" section of this document.

3. Enumeration

Once the BitFORCE device is connected to host, it will be enumerated as a "COM" port on Windows ® Based machines and as a "ttyusb" (under **/dev/**) on UNIX/LINUX based operating systems. Should the OS fail recognition, the installation of FTDI drivers must be verified.

BitFORCE SC devices are operational on start. The only exception is the MiniRig SC products which may take a few hundred milliseconds before they are fully up. This would normally not be sensed by the user or the OS, as it does not interfere with normal enumeration on the host.

Once enumerated, the device can be detected in host software by issuing a simple "Device Identification" command (later explained in this document). The device will respond with its name and version, identifying itself as a BitFORCE SC device.

4. MiniRig XLINK Chain

The MiniRig systems may incorporate more than twenty cards inside the enclosure. One of the cards which is connected to the USB will operate as XLINK chain master, and all other cards which are connected to the master card through XLINK cables, will operate as slaves.

Upon power-up, the master card will automatically initialize all devices in the chain. This will be transparent to the user or the host, even though it may take a few hundred milliseconds for the master card to be fully operational and responsive to the USB commands.

It must be noted that the data will pass each card before reaching the next card down in the chain. As a result, should one card electronically fail, all devices under it will no longer respond to chain commands. This anomaly can be simply detected by asking the master to report the number of devices in the chain.
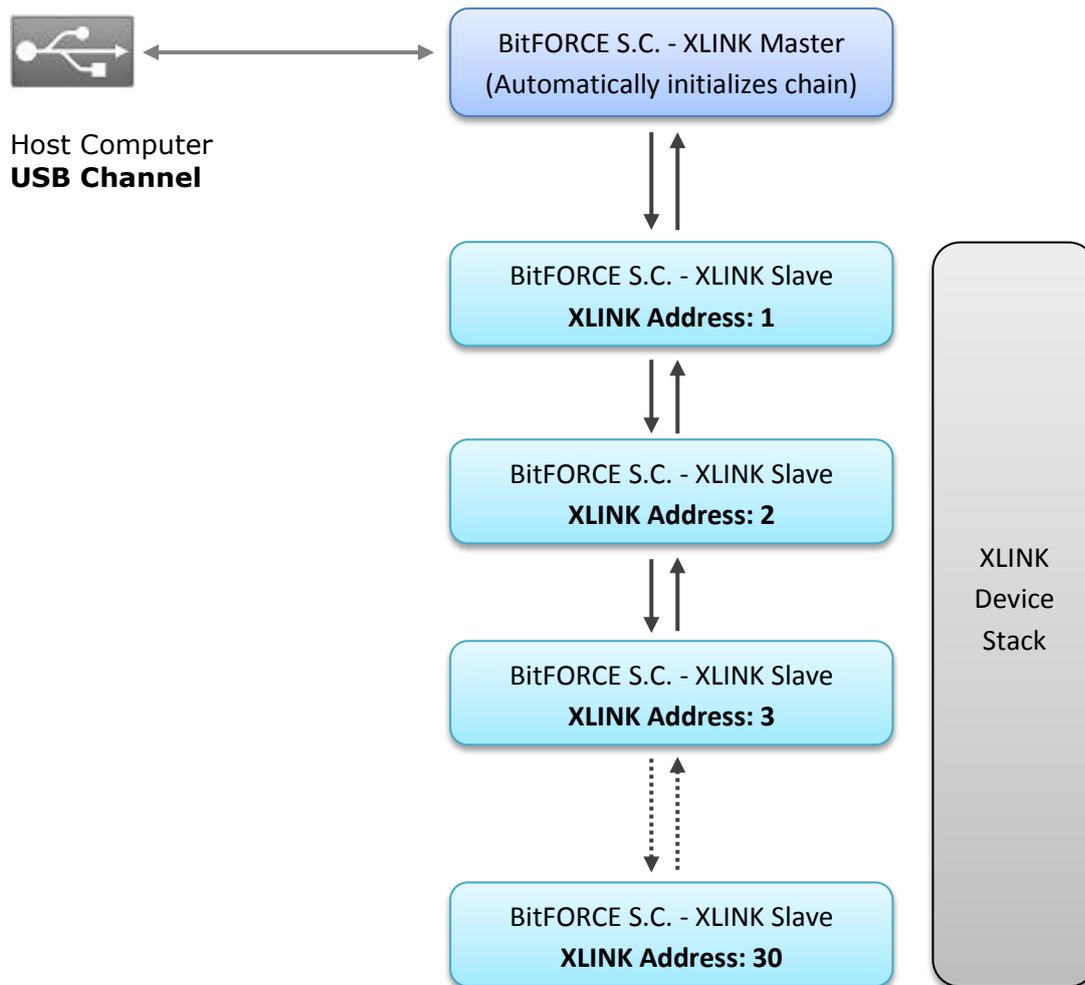
The figure above illustrates XLINK device stack, implemented in MiniRig series.

5. Sending command to devices

When sending commands, we have two possible targets: (a) Master device (including Singles and Jalapeños) and (b) Slave devices in the MiniRig chain.

(a) Commands sent to Master device (including Singles and Jalapeños) are straightforward. The three characters of the command (ZGX, ZTX, ZDX, etc.) can be sent directly to device via USB channel. No special treatment is required.

(b) Commands need to be sent to a device in chain must include a **"@XY" tag** before the actual command. The "X" is a **byte**, defining the target address (can range from 1 to 32 in binary format). "X" being "1" asks the Master to forward the command to the first device in chain after the master, "2" to the second device in chain after the master and so on.

The "Y" is a **byte,** indicating the size of the stream after this three-character header.

For instance, to send a "ZGX" command to the 4th device in the chain after the master, the command output will look like {64, 4, 3, 'Z', 'G', 'X'} (Note that "64" is the ASCII number for '@' character, "4" means the 4th device after the master, "3" means three bytes to deliver, which is three for the 'ZGX').

The following 'C' structure clarifies the structure of XLINK forward command:

```
struct DataForwardToChain
{
    unsigned __int8 Header;        // '@' by default, or numeric 64
    unsigned __int8 DeviceAddress; // Address of the device in chain
    unsigned __int8 PayloadSize;   // number of bytes to deliver
    unsigned __int8 PayloadData[]; // Bytes to be delivered, 'ZGX' for instance
                                   // Maximum allowed is 255 bytes.
};
```

6. Type of commands

Commands are either single-stage or double-stage. All the commands except job-issuing ones are single-stage; meaning device only responds to the command and goes back to IDLE stage, ready for the next command to be issued. Both single-stage and two-stage commands are completely asynchronous.

Double-stage commands (such as job-issuing) require a second input after the primary command is issued. The chart below illustrates the difference:
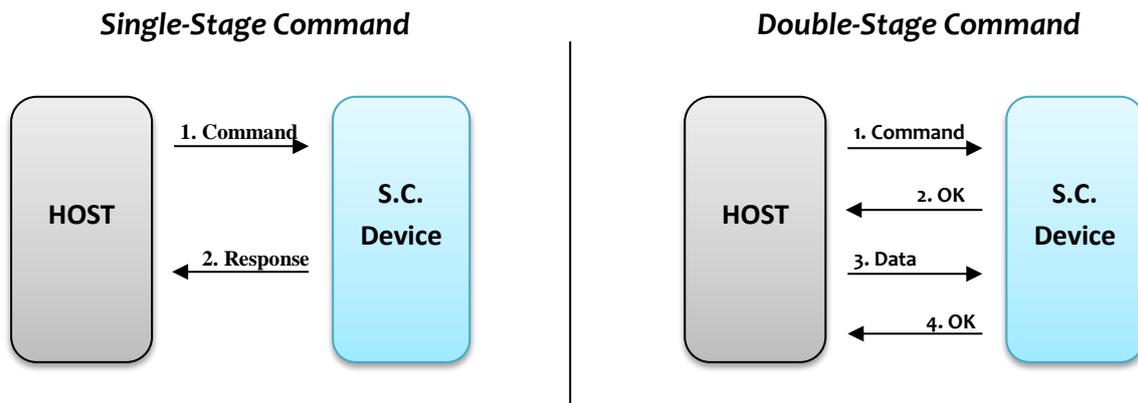
It must be noted that except "Handle Full Range Job", "Handle Custom Range Job" and "Queue Job", all commands are single-staged.

7.1 Invalid Commands & Timeouts

Should the host issue an unrecognizable command or stream of data to the device, the device will either respond with "ERROR: UKNOWN" or will not respond at all for duration of 1ms. This is to prevent device from dead-locking due to bugs in the operating software on the host.

For the double-stage commands, upon each stage, the device will await new data for duration of approximately 20**ms**. Should the host fail to provide data during this time span, the device will automatically ignore the issued command altogether and will return to IDLE state.

In any case, after a period of 1ms, the device will have returned to IDLE state and will be ready to accept new commands. This can be considered as an error-recovery strategy.

7.2 Exceptions in double-stage job-issuing commands

Job-issuing commands are all double-staged commands. Once the command is first sent to the device, the device will respond with 'OK**<LF>**' should everything be OK, waiting for the job data to be transmitted from host in the following phase of transaction.

However, there are several exceptions to this general rule:

1) 'Handle Job' command ('ZDX') along with 'Handle Job P2P' command ('ZPX') will respond with 'OK**<LF>**' in their first stage unless device is in critical-temperature recovery mode.

In this case, device will respond with:
ERR:HIGH TEMPERATURE RECOVERY**<LF>**

Command will be aborted after this response, and the device will return to IDLE state.

2) The 'Queue Job' command ('ZNX') along with 'Queue Job Pack' command ('ZWX') will return 'OK**<LF>**' in the first stage, unless:

a) The device is in critical-temperature recovery mode. In this case, device will return:
ERR:HIGH TEMPERATURE RECOVERY**<LF>**

b) The queue is full and no new job can be accepted. In this case, device will return:
ERR:QUEUE FULL**<LF>**

Command will be aborted after this response, and the device will return to IDLE state.

8. Job Queuing

BitFORCE SC firmware contains a feature which allows the host application to queue jobs inside the device. The firmware automatically issues the job to SC engines and as soon as they are finished, saves the nonces returned and proceeds with the next job in queue.

The cycle continues until all jobs in the queue are processed. It is possible to acquire the results of the jobs processed even when the queue is still working. A special 'GetStatus' command is used for this purpose, and the firmware will respond with 'Midstate', 'Data' portion of each job, along with the nonces found for that particular job (this will be addressed in detail in 'Section 9.0').

Figure below demonstrates how Job-Queue subsystem functions:
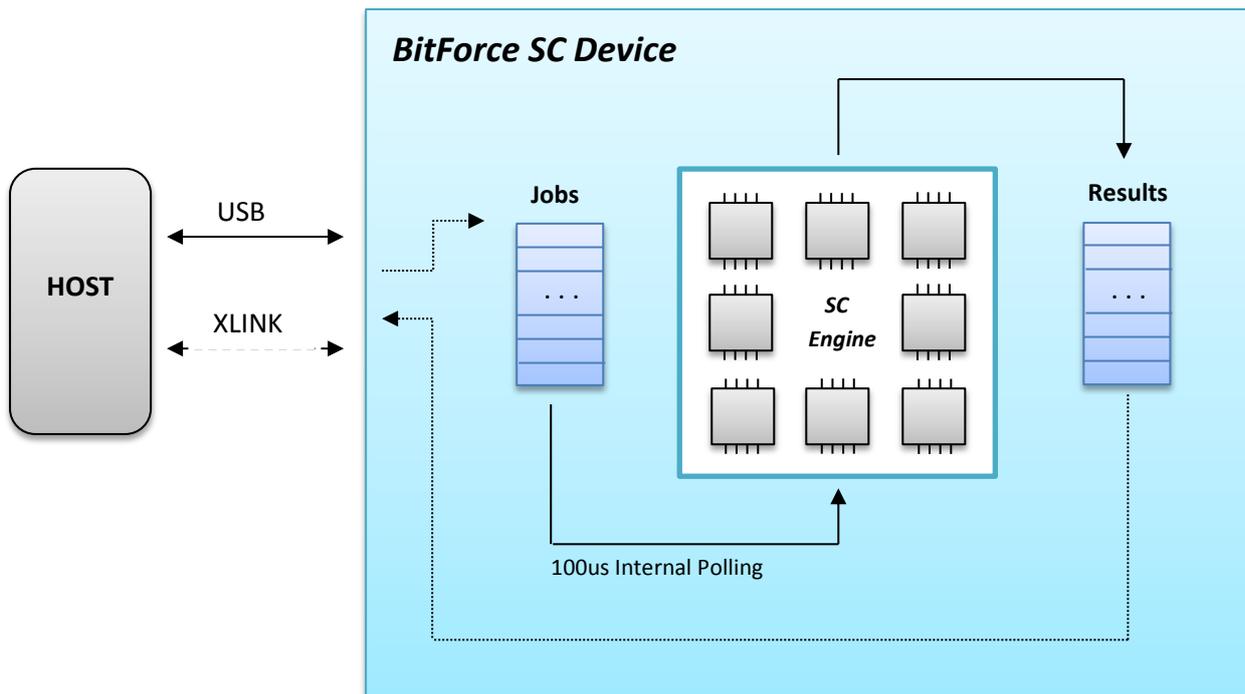
The host has the ability to issue jobs for the queue, purge the queue and read back results of the processed jobs that once existed in the queue. It must be noted that once the results are read, the 'Results' queue will be emptied to allow new results to come in. This queue will hold up to eight nonces per job for twenty jobs.

There commands are available for Job-Queue system:

- Queue Job
- Queue Job Pack
- Purge Queue
- Queue Results

These commands are examined in detail in 'Section 9.0'.

WARNING: It is important to purge job queue should the host decide not to use Job-Queue system any longer and proceed with the usual single-job calculation. Otherwise the device will finish the job requested by host and proceed with the remaining jobs in the Job-Queue.

9. BitFORCE S.C. Commands

Below you will find the list of commands issuable BitForce SC devices. Note that all of these commands can be issue to slave cards in MiniRigs as well (using the XLINK forward tag).

9.1 'ZGX' – Device Identification (Single-Stage)

This is a single stage command. Once issued to the device, it will respond with 'BitFORCE SC X.Y**<LF>**" with X and Y being numeric values defining the version of the device. For the first firmware version released, X is '1' and Y is '0'.

9.2 'ZCX' – Device Information (Single-Stage)

This is a single stage command. Once issued to the device, it will return basic information about itself:

```
DEVICE: BitFORCE SC<LF>
FIRMWARE: <Version of the firmware, starting with 1.0.0> <LF>
ENGINES: <Number of ASIC engines installed) <LF>
JOBS IN QUEUE: <Jobs in the Job-Queue waiting to be processed> <LF>
RESULTS IN QUEUE: <Number of job results awaiting to be read by host> <LF>
XLINK MODE: <MASTER or SLAVE> <LF>
XLINK PRESENT: <YES or NO> <LF>
DEVICES IN CHAIN: <Count of slave devices in XLINK chain> [CONDITIONAL]<LF>
CHAIN PRESENCE MASK: <a 32Bit hexadecimal number, see descriptions><LF>
OK <LF>
```

The 'XLINK MODE' tells whether the device is the XLINK master or a simply an XLINK slave in the chain.

The 'DEVICES IN CHAIN' field above will only appear if 'XLINK PRESENT' is valid. Should the XLINK not be present (the XLINK chip not installed on board), this field will not appear in the output.

The 'CHAIN PRESENCE MASK' field above will only appear if 'XLINK PRESENT' is valid. Each bit in this number indicates whether the corresponding device exist in chain or not. For instance Bit-1 indicates whether a device with address '1' exists in XLINK chain or not. Bit-0 and Bit-31 are reserved, always reading 0.

Please note that the end of each line is marked with an **<LF>** (i.e. **\n**).

9.3 'ZMX' – Blink (Single-Stage)

This is a single stage command. Once issued to the device, it will respond with 'OK**<LF>**' immediately and blink for approximately two seconds. This can be used for visually identifying the device.

9.4 'ZTX' – Get Voltages (Single-Stage)

This is a single stage command. Once issued to the device, it will respond with a three field text, containing the on-board voltages in millivolts for 1.0V, 3.3V and VccMain (either 12V for Single and MiniRig cards or 4.5V to 5.0V for Jalapeno devices):

```
3290,1001,12400<LF>
```

The first field is 3.3V, second is 1.0V and the third is the Vcc-Main. Please note that this line is terminated with an ‹Lf›.

9.5 'ZLX' – Get Temperatures (Single-Stage)

This is a single stage command. Once issued to the device, it will respond with detected temperature of two temperature sensors on board:

```
Temp1: 29, Temp2: 28<LF>
```

Please note that this line is finished with an Lf. This command does not exist in FPGA series of products.

9.6 'ZJX' – Get Firmware Version (Single-Stage)

This is a single stage command. Once issued to the device, it will respond with the actual firmware version of the device:

```
1.0.0<LF>
```

This is an example. Actual value will depend on the firmware version of the device.

9.7 'Z<0..5>X' – Set FAN operating mode (Single-Stage)

This is a single stage command. Once issued to the device, it will respond with 'OK**<LF>**' and modify the state of FAN system accordingly.

```
Z0X → Fan-speed lowest
Z1X → Fan-speed low
Z2X → Fan-speed medium
Z3X → Fan-speed fast
Z4X → Fan-speed fastest
Z5X → Fan-speed Auto (This mode tries to keep temperature at 60°Celcius)
```

It must be noted that state '5' means the device will set the fan speed automatically. Should the Fan-system be in any state other than '5', the device will not change the fan speed unless the components reach the high-temperature at 90° Celsius.

If this temperature is reached, the fan will start running at full speed automatically. Should device further reach 100° Celsius, the device will **stop accepting jobs** to allow the engines (and/or the VRM) to cool-down.

9.8 'ZFX' – Get Job Result Status (Single-Stage)

This is a single stage command. Once issued to the device, it will respond with the status of the engines. It can be either one of the below:

- BUSY**<LF>**
- IDLE**<LF>**
- NO-NONCE**<LF>**
- NONCE-FOUND:<XXXXXXXX>,<YYYYYYYY>,<ZZZZZZZZ>…**<LF>**

To further explain these possible responses:

-- BUSY: Device is processing a nonce

-- IDLE: Means the device has never processed any jobs and is not processing one.

-- NO-NONCE: Device has finished processing the job and no nonces were found

-- NONCE-FOUND: Means nonces have been detected. Nonces are separated with a colon and they are formatted as 32Bit hexadecimal numbers (i.e. 3ABEF01E, 2A0000B9).

A real-world example for the Nonce-Found:
`'NONCE-FOUND:0F1A9E1F,0200B4A0,89B67A00'`
**NOTE:** The response will be terminated with an <LF>.

9.9 'ZDX' – Handle Full Range Job (Double-Stage)

This is a **double stage** command. Once issued to the device, it will respond with `'OK<LF>'` (See section 7.2). This means the device is waiting for job data to be sent by the host.

If the job data is not received within **20ms**, the device assumes an anomaly exists and will respond with `'ERR: TIMEOUT<LF>'` while returning to IDLE state and waiting for a new command to be issued.

Under normal circumstances however, the host will send the job information and once received by the device (assuming the data is correctly formatted), will respond with `'OK<LF>'`. This means the device has started processing the job.

From this point, the status of engines can be determined by issuing a single 'Get Job Result Status' command (see 8.7). The diagram below illustrates this command:
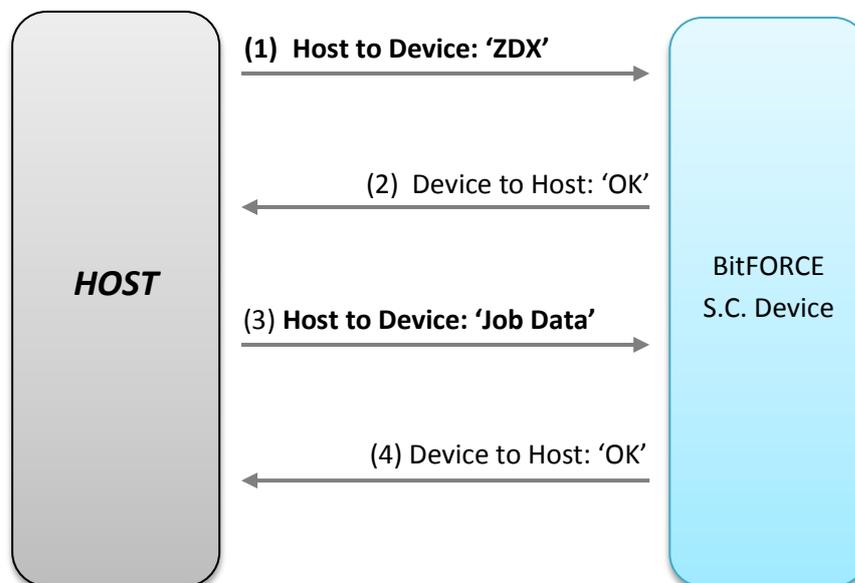

It must be noted that the device will wait **20ms** for the host to send job data. After this period, the device will send an `'ERR:TIMEOUT<LF>'` message to host and the operation will be aborted. Should the device receive data but the format is incorrect, it will respond with `'ERR:INVALID DATA<LF>'` and aborts the operation,

returning to IDLE state. Format of the Full-Range Job data is described in the 'C' structure below:

```
struct FullNonceRangeJob
{
    unsigned __int8 payloadSize;    // payload size, which is 45 (32+12+1)
    unsigned __int8 midState[32];   // 256Bits of midstate data
    unsigned __int8 blockData[12];  // 12 bytes of merkel data
    unsigned __int8 endOfBlock;     // Should be 0xAA
};
```

It must be noted that the device will not notify the host once the process has finished. The host must implement a polling strategy, asking the unit whether it has finished processing or not every 5ms or so. Job-issue commands are fully asynchronous and the device **will not be blocked** until job-processing is finished.
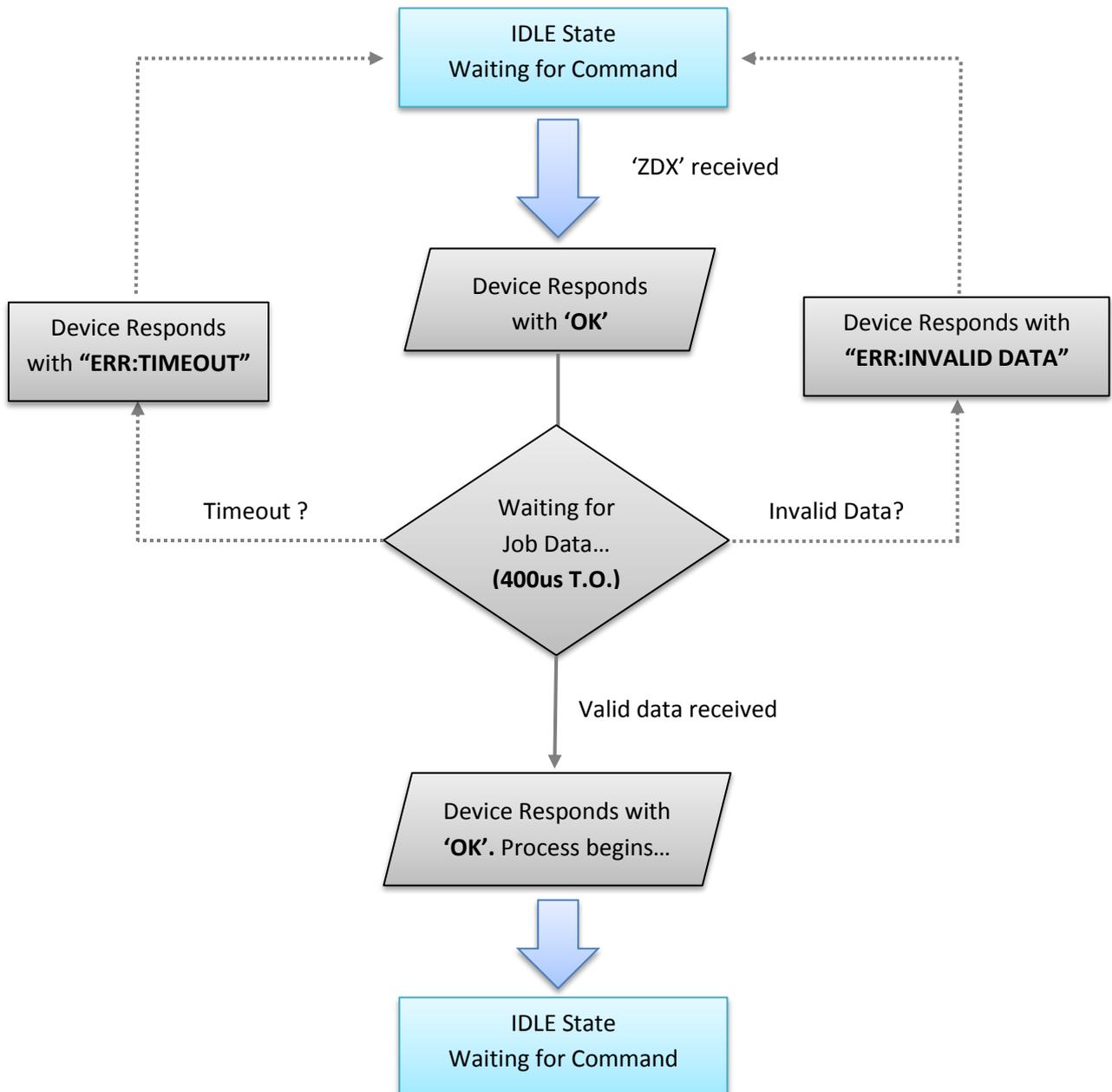
9.9 'ZPX' – Handle Custom Range Job (Double-Stage)

This is a **double stage** command. It is identical to Full-Range Job Issue (ZDX), except that the nonce-range is included in the data structure:

```
struct CustomNonceRangeJob
{
    unsigned __int8  payloadSize;   // payload length = 53 (32+12+4+4+1)
    unsigned __int8  midState[32];  // 256Bits of midstate data
    unsigned __int8  blockData[12]; // 12 bytes of merkel data
    unsigned __int32 nonceBegin;    // 4 Bytes of Nonce-Begin
    unsigned __int32 nonceEnd;      // 4 Bytes of Nonce-End
    unsigned __int8  endOfBlock;    // Signature, should be 0xAA
};
```

Once the command is successfully issued, the device will start processing the job, taking the 'nonceBegin' and 'nonceEnd' into consideration. Host can call the usual 'Get Job Result Status' command (see 8.7) to learn the status of job in process.

9.10 'ZNX' – Queue Job (Double-Stage)

This command is identical to 'Handle Full-Range Job' (ZDX), except that the job will be sent to central Job-Queue and the MCU will process it as soon as the engines finish processing the actual job (If no jobs are in process, this job is sent directly to engines for processing) and other jobs that we issued earlier.

The Job-Queue can hold a maximum of twenty jobs. On the result side, the results-queue can hold up to twenty job results (each can contain maximum of eight nonces per job). The data structure for this command is:

```
struct QueueJobStruct              // Identical to FullNonceRangeJob
{
    unsigned __int8  payloadSize;   // payload length = 45 (32+12+1)
    unsigned __int8  midState[32];  // 256Bits of midstate data
    unsigned __int8  blockData[12]; // 12 bytes of merkel data
    unsigned __int8  endOfBlock;    // Signature, should be 0xAA
};
```

Once the command is sent, the device may either respond with "**OK:QUEUED<LF>**" if the job was either queued or sent for processing, or "**ERR:QUEUE FULL<LF>**",
if the queue is already full and cannot accept a new job. If the format of the received data is incorrect, or the 'endOfBlock' is not valid, '**ERR:SIGNATURE<LF>**' will be returned by the device.

Afterwards, the device will check the engines status. If they are processing, then this job will be queued until it can be processed. If engines are IDLE, then the job will be immediately processed. To get the result of the jobs that were processed in the queue, command 'Queue Results' must be issued.

9.10.B 'ZWX' – Queue Job Pack (Double-Stage)

This command is identical to 'Queue Job' (ZNX), except that it will accept maximum five jobs at a time, passing them to central Job-Queue.

```
struct QueueJobPackStruct          // Identical to FullNonceRangeJob
{
    unsigned __int8  payloadSize;   // Byte-length of this whole
                                    // structure plus the size of
                                    // the array in bytes.
    unsigned __int8  signature;     // Signature is 0x0C1
    unsigned __int8  jobsInArray;   // Number of jobs in the array (1 to 5)
    QueueJobStruct   jobs[<count>]; // Array of jobs (QueueJobStruct)
    unsigned __int8  endOfWrapper;  // Should be 0xFE
};
```

Once the command is sent, the device may either return "**OK:QUEUED X<LF>**" if the job was either queued or sent for processing (**X** will range from 1 to 5, depending on the number of jobs that were accepted from index-zero of the array), or may respond with "**ERR:QUEUE FULL<LF>**", if the queue is already full and cannot accept jobs.

If the format of the received data is incorrect, or the 'endOfWrapper' is not valid, or any endOfBlock in job-elements in the array is not correct '**ERR:SIGNATURE<LF>**' will be returned by the device and no job will be queued.

Afterwards, the device will check the engines status. If they are processing, then this job will be queued until it can be processed. If engines are IDLE, then the job will be immediately processed. To get the result of the jobs that were processed in the queue, command 'Queue Results' must be issued.

9.11 'ZOX' – Queue Results (Single-Stage)

This command returns the data of the "Results Queue" in the Job-Queue system. The on-board controller processes the jobs found in Job-Queue one by one, and pushes the result in "Results Queue". Each entry in the queue contains the initial "MidState", the blockData bytes, along with result nonces (maximum 8 nonces).

The Midstate along with the blockData bytes can be used to identify the original work issue to the device. The format of the result is as following:

```
COUNT:<Count of entries in this report in decimal, maximum 20><LF>
[64 Mid-State],[24 blockData],[NC],[N1],[N2],[N3]..[N8]<LF>
[64 Mid-State],[24 blockData],[NC],[N1],[N2],[N3]..[N8]<LF>
.
.
.
[64 Mid-State],[24 blockData],[NC],[N1],[N2],[N3]..[N8]<LF>
OK<LF>
```

1) The '64 Mid-State' field is the 64 digits in hexadecimal, representing the midstate.

2) Next is the '24 blockData' which is 24 hexadecimal digits, representing the last twelve bytes of the block data sent to the device.

3) The [NC] is the 'Number of Nonces' in this entry. Should it be '48', the line will terminate with **<LF>** immediately after this field ([NC] ranges from '48' to '56' - which are the ASCII codes for '0' to '8').

4) The [N1] to [N8] are eight nonce results for this job. The count of nonces is indicated by the [NC] field. They are 32 bits, represented by an 8-digit hex value.

WARNING: *Once this command is issued and result is returned, the on-board controller will automatically purge the "Results Queue".*

Example:

```
COUNT:3<LF>
ffe9b1d0…0e1f,fe123498…2e9,2,876b197d,9ebc3a1f<LF>
fe109232…108a,edb3847a…983,0<LF>
de3a8e19…cd1b,acc149fd…a2f,3,ffef87ef,ab19bc1a,3f19be4d<LF>
OK<LF>
```

9.12 'ZQX' – Queue Flush

This is a single stage command. Once issued to the device, it will **ONLY** reset the Job-Queue (but will not touch the 'Results Queue' or the work in process).

The device will finish the actual job in process (should there be any), push its result in the Results-Queue will return to Idle state, waiting for a new command to arrive.

The device will always respond with 'OK**<LF>**'. Please note that the 'Results-Queue' or the actual job in process, will not be affected by this command, and the host can still access the results of jobs that were previously processed by the device.

9.13 'ZSX' – Save String (Double-Stage)

This is a double-stage command. Once issued to the device, it will return 'OK**<LF>**', waiting for the data to be sent in the second phase of the transaction.

The format of data would be:
`<PayloadLenght:`**1Byte**`><String data, maximum 255 bytes, not null terminated>`

For instance, in order to save "Butterflylabs" in device memory, the following stream must be forwarded to device:

```
Pseudo-data: <0x0D (length of 'Butterflylabs')><Butterflylabs>
Hexadecimal: 0D427574746572666C796C616273
```

```
Note: Hexadecimal is used here to represent the stream. The stream must be sent in binary
format.
```

```
Note: Binary 0 and Binary 10 are not allowed in the data-stream, as the device will use them
to mark the end of stream in flash and end-of-line in response. All other 8Bit values are
accepted.
```

At this point, should the stream have been sent correctly, the device will respond with 'SUCCESS**<LF>**'. Should it take longer that 15ms for the host to send the data stream, device may respond with 'ERR:TIMEOUT**<LF>**' and abort the operation. 'ERR:INVALID DATA**<LF>**' will be returned if stream is mal-formatted.

9.14 'ZUX' – Load String (Single-Stage)

This is a single stage command. Once issued to the device, it will respond with the string previously saved in user-space memory of the device.

The returned data will have the following format:
```
<Saved string, up to 255 characters, NOT null terminated><LF>
```

Should the memory be empty, the device will return:
```
MEMORY EMPTY<LF>
```

Revisions & Modifications

2.2.0 – Preliminary [March 2013]
→ Queue-Job structure was changed (payload-length changed to Structure-Size)
→ Added Line-Feed (**<LF>**) to QNX responses
→ Queue-Job-Pack (ZNX) command was added as a feature.
→ Queue-Results (ZOX) descriptions were modified
→ Added 'Z0X' to 'Z5X' FAN control commands
→ In the "Invalid Commands and Timeouts" section, 400us timeout was adjusted to 20ms.
→ Section '7.2 Exceptions in double-stage job-issuing commands' was added
→ Commands 'ZSX' and 'ZUX' have been added to documentation

2.1.0 – Preliminary [February 2013]
→ Name corrections were made regarding job-issuing commands
→ Corrected Payload-Length to 45 from 53.
→ Added Line-Feed (**<LF>**) to applicable responses

2.0.0 – Preliminary [December 2012]

→ Job-Queue subsystem description was added
→ Job-Queue commands were added
→ 'ZGX' corrected to 'ZDX' in the diagram on page 7.
→ All 'CrLf' were changed to 'Lf'