Flasher BitStreamer

User Manual

FPGA and CPLD Programming

Document: UM08047
Software Version: 1.32
Revision: 0
Date: November 3, 2025

A product of SEGGER Microcontroller GmbH

www.segger.com

Disclaimer

The information written in this document is assumed to be accurate without guarantee. The information in this manual is subject to change for functional or performance improvements without notice. SEGGER Microcontroller GmbH (SEGGER) assumes no responsibility for any errors or omissions in this document. SEGGER disclaims any warranties or conditions, express, implied or statutory for the fitness of the product for a particular purpose. It is your sole responsibility to evaluate the fitness of the product for any specific use.

Copyright notice

Trademarks

Names mentioned in this manual may be trademarks of their respective companies.

Brand and product names are trademarks or registered trademarks of their respective holders.

Contact address

SEGGER Microcontroller GmbH

Ecolab-Allee 5
D-40789 Monheim am Rhein

Germany

Tel. +49-2173-99312-0
Fax. +49-2173-99312-28
E-mail: support@segger.com
Internet: www.segger.com

Manual versions

This manual describes the current software version. If you find an error in the manual or a problem in the software, please report it to us and we will try to assist you as soon as possible.

Print date: November 3, 2025

Version	Revision	Date	By	Description
1.30	0	251030	PC	Initial release version.
1.20	0	250917	PC	Lead customer release version.
1.10	0	250520	PC	Internal release version.

About this document

Assumptions

This document assumes that you already have a solid knowledge of the following:

- The software tools used for building your application (assembler, linker, C compiler).
- The C programming language.
- The target processor.
- DOS command line.

If you feel that your knowledge of C is not sufficient, we recommend *The C Programming Language* by Kernighan and Ritchie (ISBN 0--13--1103628), which describes the standard in C programming and, in newer editions, also covers the ANSI C standard.

How to use this manual

This manual explains all the functions and macros that the product offers. It assumes you have a working knowledge of the C language. Knowledge of assembly programming is not required.

Typographic conventions for syntax

This manual uses the following typographic conventions:

Style	Used for
Body	Body text.
`Keyword`	Text that you enter at the command prompt or that appears on the display (that is system functions, file- or pathnames).
`Parameter`	Parameters in API functions.
`Sample`	Sample code in program examples.
`Sample comment`	Comments in program examples.
Reference	Reference to chapters, sections, tables and figures or other documents.
GUIElement	Buttons, dialog boxes, menu names, menu commands.
Emphasis	Very important sections.

Table of contents

Chapter 1

Introduction

This chapter provides a brief overview of the Flasher BitStreamer.

1.1 What is the Flasher BitStreamer?

The Flasher BitStreamer converts industry-standard device programming files into a programming solution that runs on SEGGER Flasher hardware. BitStreamer programs devices from the command line for development/testing environments and creates a package for use in Flasher production environments.

1.2 What device programmers are supported?

The Flasher BitStreamer supports the following SEGGER Flashers:

- Flasher Pro
- Flasher Pro XL
- Flasher Compact
- Flasher Portable
- Flasher ATE2

Hardware version 5 (or later) of each Flasher above is required.

1.3 What programming formats are supported?

The programming file formats supported by BitStreamer are:

- *Serial Vector Format* (SVF), and
- *Standard Test and Programming Language* (STAPL).

STAPL and SVF files are commonly produced by programmable-logic design software such as Vivado for Xilinx/AMD products and Quartus for Altera/Intel products.

SVF is standardized in SVF Serial Vector Format Specification by ASSET Intertech and STAPL is standardized in Standard Test And Programming Language (JESD71) by JEDEC.

1.4 What programmable devices are supported?

Field Programmable Gate Arrays (FPGAs) and Complex Programmable Logic Devices (CPLDs) are the natural candidates for programming using SVF and STAPL files. Because the programming file specifies exactly how to program a device, there's no need to develop a custom programming solution: BitStreamer works straight out of the box with these files.

Although FPGAs and CPLDs are the most common targets for programming, any device that can be programmed using an SVF or STAPL file is supported by Flasher BitStreamer and Flasher programmers.

As such, *any* device from *any* vendor is supported by the combination of SEGGER Flasher and Flasher BitStreamer when using SVF and STAPL. BitStreamer has been successfully tested with devices from Xilinx/AMD, Altera/Intel, Actel/Microsemi/Microchip, Lattice Semiconductor, and GOWIN Semiconductor without issue.

1.5 Related SEGGER software

The Flasher BitStreamer software was developed using the following two products from SEGGER:

- The J-Link Software Development kit (SDK), used to deploy to and run programming packages on the Flasher.
- The Flasher Software Development (SDK), used to control the target interface of the Flasher using the JTAG protocol.

If you need to program a non-standard device in production that is not covered by any SEGGER software, you can develop programming solutions for that device using the SEG-

GER SDK. The target doesn’t require a JTAG interface—SPI, I2C, and UART support is built into the Flasher SDK. If even this is not sufficient, proprietary interfaces can be supported using the flexibility delivered by the Flasher SDK and Flasher hardware.

1.6 Delivery and installation

BitStreamer is delivered as a 64-bit Windows application, `FlasherBitStreamer.exe`, that runs on Windows 10 and later (Intel or Arm silicon). Copy the BitStreamer application to the most convenient place for your workflow.

Chapter 2

Using BitStreamer

2.1 Programming devices in testing

The following sections describe the process of programming a device in a development or test enviroment rather than in a production environment.

2.1.1 Programming SVF files

SVF files are plain text files containing a list of instructions that describe transactions to be performed by a JTAG interface. These files typically use the file extension ".svf".

A device can be programmed from an SVF file by invoking BitStreamer with the `--erase` and `--run` options, along with the appropriate SVF file. The `--erase` option erases all existing content on the Flasher, and the `--run` option instructs the Flasher to run programming immediately after deployment.

Example

```
C:> FlasherBitStreamer.exe --erase --run Altera_5M570ZN_Board_18MHz.svf

SEGGER Flasher BitStreamer V1.20 compiled Sep 17 2025 02:54:57
Copyright (c) 2024-2025 SEGGER Microcontroller GmbH    www.segger.com

Deploying to Flasher:
  Altera_5M570ZN_Board_18MHz_001.adf     82007 bytes   0.110 s, 726 KB/sec
  Altera_5M570ZN_Board_18MHz_002.adf     80073 bytes   0.081 s, 965 KB/sec
  Altera_5M570ZN_Board_18MHz_003.adf     49322 bytes   0.034 s, 1396 KB/sec
  Altera_5M570ZN_Board_18MHz_004.adf     46954 bytes   0.030 s, 1546 KB/sec
  Altera_5M570ZN_Board_18MHz.pex          1676 bytes   0.019 s, 86 KB/sec

Executing:
  Successful execution

Streaming complete: 0 errors.

C:> _
```

Should there be a device programming error, BitStreamer will report a failure.

2.1.2 Programming STAPL files

STAPL files are plain text files containing a program that controls transactions performed by a JTAG interface. These files typically use the file extension ".jam" or ".stp".

A device can be programmed from a STAPL file by invoking BitStreamer with the `--erase` and `--run` options, along with the appropriate STAPL file. The `--erase` option erases all existing content on the Flasher, and the `--run` option instructs the Flasher to run programming immediately after deployment.

Example

```
C:> FlasherBitStreamer.exe --erase --run Altera_5M570ZN_Board.jam

SEGGER Flasher BitStreamer V1.20 compiled Sep 23 2025 00:04:01
Copyright (c) 2024-2025 SEGGER Microcontroller GmbH    www.segger.com

Deploying to Flasher:
  Altera_5M570ZN_Board_001.adf     13824 bytes   0.085 s, 158 KB/sec
  Altera_5M570ZN_Board.pex         26800 bytes   0.062 s, 420 KB/sec

Executing:
  Device #1 Silicon ID is ALTERA10(04)
  erasing MAXII device(s)...
  erasing MAXII UFM block...
  erasing MAXII CFM block...
  programming CFM block...
  programming UFM block...
  verifying CFM block...
  verifying UFM block...
```



```
  DONE
  Successful execution

Streaming complete: 0 errors.

C:> _
```


2.2 Preparing for production programming

BitStreamer is able to create file packages from SVF and STAPL inputs to be used with the Flasher Deployer application.

2.2.1 Preparing SVF files

To create a file package for Flasher Deployer, run BitStreamer passing only the name of the SVF or STAPL file:

```
C:> FlasherBitStreamer.exe Altera_5M570ZN_Board_18MHz.svf

SEGGER Flasher BitStreamer V1.20 compiled Sep 23 2025 00:04:01
Copyright (c) 2024-2025 SEGGER Microcontroller GmbH    www.segger.com

Streaming complete: 0 errors.

C:> _
```

BitStreamer packages all the files required to program the device (or devices) into a Zip file based on the input file name: in this case, `Altera_5M570ZN_Board_18MHz.fzip` is written to disk. This file can be dragged onto the Flasher Deployer to deploy to a fleet of Flashers.

The **-v** command-line option provides some more detail regarding the files written to disk:

```
C:> FlasherBitStreamer.exe -v Altera_5M570ZN_Board_18MHz.svf

SEGGER Flasher BitStreamer V1.20 compiled Sep 23 2025 00:04:01
Copyright (c) 2024-2025 SEGGER Microcontroller GmbH    www.segger.com

Files written to disk:
  Altera_5M570ZN_Board_18MHz_001.adf     82007 bytes   TMS bitstream (  1.61% of 5094623)
  Altera_5M570ZN_Board_18MHz_002.adf     80073 bytes   TDI bitstream (  1.57% of 5094623)
  Altera_5M570ZN_Board_18MHz_003.adf     49322 bytes   TDO bitstream (  0.97% of 5094623)
  Altera_5M570ZN_Board_18MHz_004.adf     46954 bytes   MSK bitstream (  0.92% of 5094623)
  Altera_5M570ZN_Board_18MHz.pex          1676 bytes   App executable
  Altera_5M570ZN_Board_18MHz.fzip       260996 bytes   Flasher Deployer package

Streaming complete: 0 errors.

C:> _
```

The file package contains five files: four are the bitstreams to be sent to the JTAG interface, and one is the programming application responsible for transferring those bitstreams to the JTAG interface using Flasher hardware. It's not required to understand what these files contain or how they perform the programming algorithm, only that all files are required for successful programming.

2.2.2 Preparing STAPL files

Preparing STAPL files for production programming is equally simple:

```
C:> FlasherBitStreamer.exe Altera_5M570ZN_Board.jam

SEGGER Flasher BitStreamer V1.20 compiled Sep 23 2025 00:04:01
Copyright (c) 2024-2025 SEGGER Microcontroller GmbH    www.segger.com

Files written to disk:
  Altera_5M570ZN_Board_001.adf     13824 bytes   Data for 'A99'
  Altera_5M570ZN_Board.pex         26800 bytes   App executable
  Altera_5M570ZN_Board.fzip        40962 bytes   Flasher Deployer package

Actions exported:
  PROGRAM

Streaming complete: 0 errors.

C:> _
```

2.3 Customized programming

The following sections provide some additional information to successfully configure or program devices.

2.3.1 Programming frequencies

Both SVF and STAPL scripts are able to select the maximum frequency of the TCK signal using a **FREQUENCY** statement. From this, BitStreamer will correctly configure the JTAG interface to this frequency and not clock it faster.

When an SVF or STAPL file does not specify a frequency using **FREQUENCY**, BitStreamer uses a default frequency of 1 MHz. It is possible to override this using the --freq command-line option. For example, to set the default frequency to 25 MHz, use:

```
C:> FlasherBitStreamer.exe --freq 25000000 Altera_5M570ZN_Board.jam
```

Generally, programming completes faster using higher-frequency clocking, but please be aware of the limitations of devices attached to the the JTAG chain and do not clock the interface faster than the slowest device on the chain: device datasheets usually provide clocking information in “device parameters” sections.

2.3.2 STAPL actions

STAPL programs provide a number of *actions* corresponding to operations that can be performed on the device. BitStreamer supports immediate execution of these actions and creating Flasher file packages for production environments.

2.3.2.1 Standard actions

The STAPL specification documents the following *standard actions*:

Action name	Description
CHECKCHAIN	Verify continuity of IEEE 1149.1 scan chain
READ_IDCODE	Read the IEEE 1149.1 IDCODE and export it
READ_USERCODE	Read the IEEE 1149.1 USERCODE and export it
READ_UES	Read the UESCODE and export it
ERASE	Bulk-erase device
BLANK_CHECK	Check device is erased
PROGRAM	Program device
VERIFY	Verify programming of device
READ	Read programming data from device
CHECKSUM	Calculate fuse checksum of the device
SECURE	Set the security bit of the device
QUERY_SECURITY	Check whether the security bit is set
TEST	Perform a test.

Although these are documented in the specification, vendors define their own actions beyond this and even deviate from the standard naming scheme. For instance, Altera uses **BLANKCHECK** rather than **BLANK_CHECK**, and extends actions with **CHECK_IDCODE**.

2.3.2.2 Selecting actions

BitStreamer uses the following decision process to select an action:

- If there is exactly one **ACTION** statement in the STAPL program, that action is selected.

- If there are multiple **ACTION** statements in the STAPL program, and one of them is named **PROGRAM**, the **PROGRAM** action is selected.
- In all other cases, BitStreamer will issue an error diagnostic and require a selection to be made using the **--action** command-line option.

2.3.2.2.1 Example with a single action

The STAPL script here exports a single action, **PROBE_CHAIN**, which is automatically handled:

```
C:> FlasherBitStreamer.exe -v ProbeChain.jam

SEGGER Flasher BitStreamer V1.20 compiled Sep 23 2025 00:04:01
Copyright (c) 2024-2025 SEGGER Microcontroller GmbH    www.segger.com

Files written to disk:
  ProbeChain.pex     13432 bytes   App executable
  ProbeChain.fzip    13572 bytes   Flasher Deployer package

Actions exported:
  PROBE_CHAIN

Streaming complete: 0 errors.

C:> _
```

BitStreamer indicates the action selected for execution is **PROBE_CHAIN**.

2.3.2.2.2 Example with a PROGRAM action

The STAPL script here exports a multiple actions, but a **PROGRAM** action is among them:

```
C:> FlasherBitStreamer.exe -v Altera_5M570ZN_DevBoard.jam

SEGGER Flasher BitStreamer V1.20 compiled Sep 23 2025 00:04:01
Copyright (c) 2024-2025 SEGGER Microcontroller GmbH    www.segger.com

Files written to disk:
  Altera_5M570ZN_MAXV_DevBoard_001.adf     13824 bytes   Data for 'A99'
  Altera_5M570ZN_MAXV_DevBoard.pex         26800 bytes   App executable
  Altera_5M570ZN_MAXV_DevBoard.fzip        40962 bytes   Flasher Deployer package

Actions exported:
  PROGRAM

Diagnostics:
  Altera_5M570ZN_DevBoard.jam:33: warning: multiple actions declared; selecting 'PROGRAM'
 action
  remark: use command-line option '-a' to override this choice with one of the following
 actions:
  remark:   BLANKCHECK
  remark:   CHECK_IDCODE
  remark:   ERASE
  remark:   PROGRAM
  remark:   READ_USERCODE
  remark:   VERIFY

Streaming complete: 0 errors, 1 warning, 7 remarks.

C:> _
```

BitStreamer indicates the action selected for execution is **PROGRAM** even though other actions exist.

2.3.2.2.3 Example of action selection

Selection of an action is possible using the `-a` or `--action` command-line option. To override the previous example's selection of **PROGRAM**, provide the command-line option **-aERASE**:

```
C:> FlasherBitStreamer.exe -v -aERASE Altera_5M570ZN_DevBoard.jam

SEGGER Flasher BitStreamer V1.20 compiled Sep 23 2025 14:26:14
```

```
Copyright (c) 2024-2025 SEGGER Microcontroller GmbH    www.segger.com

Files written to disk:
  Altera_5M570ZN_DevBoard.pex     19148 bytes   App executable
  Altera_5M570ZN_DevBoard.fzip    19324 bytes   Flasher Deployer package

Actions exported:
  ERASE

Streaming complete: 0 errors.

C:> _
```

BitStreamer indicates the action selected for execution is **ERASE**, overriding the default selection of **PROGRAM**.

Note that STAPL ignores letter case in identifiers, so the action can also be written in lowercase or mixed case, for instance **-aerase** and **-aErase** both work to select the **ERASE** action.

Chapter 3

Extended STAPL

This chapter describes the enhancements to the STAPL language brought by SEGGER Extended STAPL.

3.1 Introduction

SEGGER Extended STAPL is a set of extensions to the JEDEC standard STAPL specification. These extensions makes writing hand-written STAPL programs easier to construct and maintain.

The enhancements are:

- Adding a preprocesor to further enhance the language the way C and C++ programmers are familiar with.
- Introduction of structured programming capabilities, such as loops and multi-way branching.
- Extending expressions with vector replication and vector concatenation.
- Adding simple formatting capabilities when formatting output.

The following sections describe these enhancements.

3.2 Preprocessor

SEGGER Extended STAPL provides a standard C99-style preprocessor to make writing STAPL applications simpler. It, however, also features some extensions to the C90 standard.

3.2.1 #warning control

The **#warning** preprocessor directive issues a warning in the same manner that **#error** issues an error.

3.2.2 #remark control

The **#remark** preprocessor directive issues a remark in the same manner that **#error** issues an error.

3.3 Statements

SEGGER Extended STAPL provides looping and multiway branching capabilities that are standard in many programming languages.

3.3.1 REPEAT statement

Syntax

REPEAT;
    statement…
UNTIL *expr* **;**

Description

Repeat execution of each statement between **REPEAT** and **UNTIL** until the Boolean expression *expr* evaluates to true. The loop executes at least once.

Example

```
BOOLEAN State[32];

PRINT "Waiting for READY status";
'
REPEAT;
  PRINT "Scan READY status";
  DRSCAN 32, $00000000, CAPTURE State[];
UNTIL State[0] == 1;
'
PRINT "READY asserted";
```


3.3.2 WHILE statement

Syntax

```
WHILE expr ;
    statement…
WEND;
```

Description

Repeat execution of each statement between **WHILE** and **WEND** whilst *expr* evaluates to true. The loop can iterate zero times if *expr* initially evaluates to false.

Example

```
INTEGER Data[4] = 11, 22, 33, 44;
BOOLEAN State[32];

I = 4;
WHILE I > 0;
  I = I - 1;
  DRSCAN 32, BOOL(Data[I]);
WEND;
```


3.3.3 CASE statement

Syntax

```
CASE value OF
    { WHEN const-expr : statement… }
    [ WHEN ELSE: statement… ]
ENDCASE;
```

Description

The integer expression *value* is evaluated and matched against each value specified in **WHEN** clauses. Each **WHEN** selection value must be unique within the **CASE** statement. If *value* matches any **WHEN** clause value, the statements selected by that clause are executed and control passes to the statement following the matching **ENDCASE**. If *value* does not match any **WHEN** clause, the statements selected by **WHEN ELSE**, if it exists, are executed and control passes to the statement following the matching **ENDCASE**.

Example

```
INTEGER I;

FOR I = 0 TO 4;
  CASE I OF
    WHEN 1:    PRINT "ONE... "; PRINT;
    WHEN 2:    PRINT "TWO... "; PRINT;
    WHEN ELSE: PRINT "("; I; ") ";
  ENDCASE;
NEXT I;
```


3.3.4 BEGIN statement

Syntax

```
BEGIN
    statement…
END
```

Description

The **BEGIN-END** construct enables multiple statements to be grouped as one creating a *compound statement*. This is particularly useful when combined with an **IF** statement to execute multiple statements when a condition is true, eliminating the use of **GOTO** or multiple **IF** statements typically used to synthesize this simple construct.

Example

```
INTEGER I;

FOR I = 0 TO 4;
  IF 1 < I && I < 3 THEN BEGIN
    PRINT I; "...has to be two";
    PRINT;
  END;
NEXT I;
```


3.4 Expressions

SEGGER Extended STAPL provides some additional facilities that make writing STAPL applications easier.

3.4.1 Binary and hexadecimal constants

Syntax

0b(*0-1*)…
0x(*0-9a-fA-F*)…

Description

Standard C90-style hexadecimal and C++14-style binary constants are suppoted as a convenience.

Note that binary and hexadecimal constants can be synthesized in JEDEC standard STAPL using Boolean arrays and the **INT()** function as shown below.

Example

```
INTEGER X;

X = 0b1101;  // equivalent to X = INT(#1101), i.e. X=13
X = 0x1FFE;  // equivalent to X = INT($1FFE), i.e. X=8190
```

3.4.2 Vector concatenation

Syntax

expr-a **+** *expr-b*

Description

Concatenates the Boolean array expression *expr-a* and the Boolean array expression *expr-b* to create a new Boolean array. Both *expr-a* and *expr-b* must be constant Boolean arrays.

Example

```
DRSCAN 17, $FFFF + #0;
```

3.4.3 Vector replication

Syntax

expr-a ***** *expr-b*

Description

Replicates the Boolean array expression *expr-b*, concatenating each replication, *expr-a* times to create a new Boolean array. *expr-a* must be an integer constant expression and *expr-b* must be a constant Boolean array.

Example

```
DRSCAN 1024, 1024 * #1;
```


3.5 Formatting

3.5.1 HEX$

Syntax

HEX$(*expr* **)**

Description

Within a **PRINT** statement, **HEX$()** formats the value of the integer expression *expr* as an 8-nibble hexadecimal value prefixed by a dollar sign.

Example

```
INTEGER V;

V = 100;
PRINT "V=", HEX$(V)
```

Prints:

```
V=$00000064
```


3.6 Example Extended STAPL application

The code below probes the JTAG chain and prints the ID codes of the devices that it finds. It demonstrates the features of SEGGER Extended STAPL to achieve this, using the preprocessor, Boolean vector replication and concatenation, structed looping, multiway branching, and simplified formatting.

```
' ********************************************************************
' *                   (c) SEGGER Microcontroller GmbH                *
' *                        The Embedded Experts                      *
' *                           www.segger.com                         *
' ********************************************************************
'
' -------------------------- END-OF-HEADER -----------------------------
'
' Purpose     : Scan JTAG chain for devices.
' Notes       : This file is written using SEGGER Extended STAPL.
'
'
' This can be altered from the BitStreamer command line using
' -DMAX_SCAN_SIZE=n, where n is an integer constant.
'
#if !defined(MAX_SCAN_SIZE)
  #define MAX_SCAN_SIZE   512
#endif

'
' Syntactic sugar to make device database easier to construct.
'
#define DEV(ID, NAME)     WHEN ID: PRINT "  DEVICE #", I, "  IDCODE ", HEX$(ID), "  ", NAME

'
' This file has a single action that identifies devices on
' the JTAG chain.
'
ACTION PROBE_CHAIN = DO_PROBE_CHAIN;

'
' This is the procedure that probes the JTAG chain for devices.
'
PROCEDURE DO_PROBE_CHAIN;
  BOOLEAN X[2*MAX_SCAN_SIZE];
  BOOLEAN IDCODE[32];
  INTEGER IDCNT;
  INTEGER I;
  '
  ' Sign on.
  '
  PRINT;
  PRINT "SEGGER JTAG Chain Scanner run on ", __DATE__, " ", __TIME__;
  PRINT "Copyright (c) 2024-2025 SEGGER Microcontroller GmbH    www.segger.com";
  PRINT;
  '
  ' Go to known JTAG reset state.
  '
  STATE RESET;
  '
  ' Scan all-ones into IR loading the BYPASS instruction to all devices.
  '
  IRSCAN MAX_SCAN_SIZE, MAX_SCAN_SIZE * #1;
  '
  ' Scan in all ones and zeroes and capture the output.
  '
  DRSCAN 2*MAX_SCAN_SIZE, MAX_SCAN_SIZE * #1 + MAX_SCAN_SIZE * #0, CAPTURE X[];
  '
  ' BYPASS is a single-bit register, see how many devices
  ' were bypassed by counting the number of zero bits returned.
  '
  FOR I = MAX_SCAN_SIZE TO 2*MAX_SCAN_SIZE-1;
    IF X[I] == 1 THEN BEGIN
      IDCNT = I - MAX_SCAN_SIZE;
      I = 2*MAX_SCAN_SIZE;  ' force exit of FOR
    END;
  NEXT I;
```

```
  '
  ' Dump number of devices in the chain.
  '
  PRINT "Chain:";
  IF IDCNT == 0 THEN BEGIN
    PRINT "  No devices responded on the JTAG chain";
    GOTO Done;
  END;
  '
  PRINT "  Detected ", IDCNT, " devices in the JTAG chain";
  PRINT;
  '
  ' Go to JTAG reset state: all devices in the chain load the
  ' IDCODE instruction into their instruction register.
  '
  STATE RESET;
  '
  ' Capture 32 bit IDCODE per device in theJTAG chain.
  '
  DRSCAN 32*IDCNT, $0, CAPTURE X[32*IDCNT-1..0];
  '
  ' Dump IDCODEs.
  '
  PRINT "Detected devices:";
  FOR I = 1 TO IDCNT;
    INTEGER ID;
    '
    ' Extract device's 32-bit IDCODE from the response.
    '
    ID = INT(X[32*I-1..32*I-32]);
    '
    ' Decode IDCODE to human-readable form.
    '
    CASE ID OF
      '
      ' Arm devices
      '
      DEV($4BA00477, "Arm CoreSight JTAG-DP");
      '
      ' Altera MAX II devices
      '   https://cdrdv2-public.intel.com/655094/max2_mii5v1.pdf
      '
      DEV($020A10DD, "Altera Max II EPM240(G)");
      DEV($020A20DD, "Altera Max II EPM570(G)");
      '
      ' Altera MAX V devices
      '   https://cdrdv2-public.intel.com/654928/max5_handbook.pdf table 6-3
      '
      DEV($020A50DD, "Altera 5M40Z / 5M80Z / 5M160Z / 5M240Z");
      DEV($020A60DD, "Altera 5M240Z / 5M570Z");
      DEV($020A30DD, "Altera 5M1270Z");
      DEV($020A40DD, "Altera 5M1270Z / 5M2210Z");
      '
      ' Altera Cyclone II devices
      '   https://cdrdv2-public.intel.com/654376/cyc2_cii5v1.pdf
      '
      DEV($020B10DD, "Altera Cyclone II EP2C5");
      DEV($020B20DD, "Altera Cyclone II EP2C8");
      DEV($020B30DD, "Altera Cyclone II EP2C15/20");
      DEV($020B40DD, "Altera Cyclone II EP2C35");
      DEV($020B50DD, "Altera Cyclone II EP2C50");
      DEV($020B60DD, "Altera Cyclone II EP2C70");
      '
      ' Altera Cyclone V devices
      '   https://www.intel.com/content/www/us/en/docs/programmable/683375/current/
idcode.html
      '
      DEV($02D120DD, "Altera Cyclone V SE A5");
      DEV($02D020DD, "Altera Cyclone V SE A6");
      '
      ' Altera MAX 10 devices
      '    https://www.intel.com/content/www/us/en/docs/programmable/683210/current/jtag-
idcode.html
      '
      DEV($031810DD, "Altera 10M02 (not U324)");
      DEV($0319A0DD, "Altera 10M02 (U324)");
```



```
      '
      ' GOWIN devices
      '   https://cdn.gowinsemi.com.cn/TN653E.pdf
      '   https://cdn.gowinsemi.com.cn/UG290E.pdf
      '
      DEV($0900281B, "GOWIN GW1N-1");
      DEV($0900381B, "GOWIN GW1N-1S");
      DEV($0100681B, "GOWIN GW1NZ-1");
      DEV($0120681B, "GOWIN GW1N(R/Z)-2/2B/2C");
      DEV($0100381B, "GOWIN GW1N(R)-4");
      DEV($1100381B, "GOWIN GW1N(R)-4B/4D");
      DEV($0100881B, "GOWIN GW1NS-4");
      DEV($0100981B, "GOWIN GW1NS(ER)-4C");
      DEV($1100581B, "GOWIN GW1N(R)-9");
      DEV($1100481B, "GOWIN GW1N(R)-9C");
      DEV($0000081B, "GOWIN GW2A(R)-18/18C");
      DEV($0000281B, "GOWIN GW2A-55/55C");
      '
      ' Generic device that isn't decoded.
      '
      WHEN ELSE: PRINT "  DEVICE #", I, "  IDCODE ", HEX$(ID), "  -- unknown --";
      '
    ENDCASE;
  NEXT I;
  '
  PRINT;
  PRINT "Chain detection complete";
  PRINT;
Done:
  '
ENDPROC;
```


Chapter 4

Implementation details

The following sections deal with the implementation of SVF and STAPL in the Flasher.

4.1 STAPL implementation

4.1.1 PRINT implementation

The JEDEC STAPL specification does not document the precise semantics of the **PRINT** statement and describes it only as a debugging tool. However, STAPL applications output by design tools generally use the **PRINT** statement to indicate progress of the application through programming or configuration steps and to report any errors.

The SEGGER implementation of the **PRINT** statements follows that of Altera Quartus tools with some extensions. The following sections describe the SEGGER implementation of the **PRINT** statement.

4.1.1.1 Rendering of printed output

PRINT statements output text using the function `SYS_Reportf()` of the Flasher SDK. When BitStreamer is invoked with the **--run** command-line option, such output is captured and written to the standard output stream preceded by two spaces.

Take, for example, the following program:

```
ACTION APP = DO_HELLO;

PROCEDURE DO_HELLO;
  PRINT "Hello, world!";
ENDPROC;
```

This can be run on the Flasher using:

```
C:> FlasherBitStreamer.exe --erase --run Hello.jam

SEGGER Flasher BitStreamer V1.20 compiled Sep 17 2025 02:54:57
Copyright (c) 2024-2025 SEGGER Microcontroller GmbH    www.segger.com

Deploying to Flasher:
  Hello.pex       584 bytes   0.075 s, 7 KB/sec

Executing:
  Hello, world!
  Successful execution

Streaming complete: 0 errors.

C:> _
```

When run in verbose mode using the **-v** command-line option, the entire interaction with the Flasher is displayed on standard output:

```
C:> FlasherBitStreamer.exe -v --erase --run Hello.jam

SEGGER Flasher BitStreamer V1.20 compiled Sep 17 2025 02:54:57
Copyright (c) 2024-2025 SEGGER Microcontroller GmbH    www.segger.com

Deploying to Flasher:
  Hello.pex       584 bytes   0.075 s, 7 KB/sec

Executing:
  #ACK
  #STATUS:EXECUTING
  #INFO: [App] Hello, world!
  #INFO:App main() returned 0.
  #OK (Total 0.000s)
  Successful execution

Streaming complete: 0 errors.

C:> _
```


The lines output above are:

#ACK
Acknowledgement of the Flasher **#AUTO** command issued by BitStreamer to initiate execution of the application.

#STATUS:EXECUTING
Indication of the Flasher status, in this case execution of the application has commenced.

#INFO: [App] Hello, world!
Information relating to the current process, in this case a message output by the executing application.

#INFO:App main() returned 0.
Information relating to the execution of the application, the application exited with a zero (success) exit status.

#OK (Total 0.000s)
Indicates that the application ran with successful termination and provides the time elapsed between application startup and termination.

Note that application output is written to the standard output stream and all other messages are written to the standard error stream, therefore it is possible to capture the application's output verbatim to a file or to discard unwanted messages:

```
C:> FlasherBitStreamer.exe --erase --run --no-indent-output Hello.jam 2>NUL
Hello, world!

C:> _
```

4.1.1.2 Printing multiple items

A **PRINT** statement containing more than one expression emits no spacing between the values printed.

Example

```
PRINT 1, 2, 3;
```

Output:

```
123
```


4.1.2 EXPORT implementation

The JEDEC STAPL specification does not document the precise semantics of the **EXPORT** statement or what can be exported.

4.1.2.1 Exporting integers and Boolean arrays

SEGGER's STAPL implementation supports exporting integer variables and Boolean arrays of up to 32 elements.

Integer variables are output in decimal and Boolean arrays are output in hexadecimal.

Take, for example, the following program:

```
ACTION APP = DO_EXPORT;

PROCEDURE DO_EXPORT;
  INTEGER X = 123;
  BOOLEAN A[32] = $DEAD + $BEEF;

  EXPORT "X_VALUE", X;
  EXPORT "A_VALUE", A[];
ENDPROC;
```

This can be run on the Flasher using:

```
C:> FlasherBitStreamer.exe --erase --run Export.jam

SEGGER Flasher BitStreamer V1.20 compiled Sep 17 2025 02:54:57
Copyright (c) 2024-2025 SEGGER Microcontroller GmbH    www.segger.com

Deploying to Flasher:
  Export.pex       748 bytes   0.074 s, 9 KB/sec

Executing:
  X_VALUE=123
  A_VALUE=0xDEADBEEF
  Successful execution

Streaming complete: 0 errors.

C:> _
```

EXPORT statements output text using the function `SYS_Reportf()`.

4.1.3 EXIT implementation

The JEDEC STAPL specification documents a number of standard exit codes with values 0 through 17. In particular, the zero exit code is the only exit code that denotes success. Altera extends this range with additional (nonstandard) error indications.

SEGGER translates any nonzero exit code into a negative value. That is, an exit code of 10 is translated to −10 and an exit code of −10 (if programmed using an explicit **EXIT -10**) remains −10.

STAPL exit codes are displayed by BitStreamer when the application terminates. Take, for example, the following program that intends to fail:

```
ACTION APP = DO_EXIT;

PROCEDURE DO_EXIT;
  EXIT 10;
ENDPROC;
```

This can be run on the Flasher using:

```
C:> FlasherBitStreamer.exe --erase --run ExitError.jam

SEGGER Flasher BitStreamer V1.20 compiled Sep 24 2025 16:50:44
Copyright (c) 2024-2025 SEGGER Microcontroller GmbH    www.segger.com

Deploying to Flasher:
  ExitError.pex       536 bytes   0.085 s, 6 KB/sec

Executing:
  Failed execution

fatal: programming failed

C:> _
```

To see the exit code returned by the STAPL application, run BitStreamer in verbose mode:

```
C:> FlasherBitStreamer.exe -v --erase --run ExitError.jam

SEGGER Flasher BitStreamer V1.20 compiled Sep 24 2025 16:50:44
Copyright (c) 2024-2025 SEGGER Microcontroller GmbH    www.segger.com

Deploying to Flasher:
  ExitError.pex       536 bytes   0.085 s, 6 KB/sec

Actions exported:
  APP

Executing:
  #ACK
  #STATUS:EXECUTING
  #ERR255:App main() returned error (-10).
  Failed execution

fatal: programming failed

C:> _
```


4.2 Common script problems

SEGGER's implementation of SVF and STAPL detects a number of issues in scripts generated by vendor design tools. The following sections describe the issues which can be encountered.

4.2.1 GOWIN tooling

The GOWIN design tools do not correctly output SVF files that conform to the syntax described by the SVF specification. In particular, the **MASK** and **SMASK** keywords sometimes appear in the incorrect order. This issue is diagnosed by BitStreamer, an example of which is:

```
C:> FlasherBitStreamer.exe BEA_GW1NS-4_CustomBoard.svf

SEGGER Flasher BitStreamer V1.20 compiled Sep 24 2025 12:58:12
Copyright (c) 2024-2025 SEGGER Microcontroller GmbH    www.segger.com

Diagnostics:
  BEA_GW1NS-4_CustomBoard.svf:19: warning: 'MASK' should come before 'SMASK'
  BEA_GW1NS-4_CustomBoard.svf:42: warning: 'MASK' should come before 'SMASK'
  BEA_GW1NS-4_CustomBoard.svf:67: warning: 'MASK' should come before 'SMASK'
  BEA_GW1NS-4_CustomBoard.svf:116394: warning: 'MASK' should come before 'SMASK'

Streaming complete: 0 errors, 4 warnings.

C:> _
```

These warnings can be safely ignored or suppressed using the command-line option **--no-warnings**.

4.2.2 Altera tooling

The Altera Quartus tools generate unstructured STAPL code that leads to suboptimal execution. Whilst not an error, it occurs frequently when dealing with STAPL applications output by Quartus. This issue is diagnosed by BitStreamer, an example of which is:

```
C:> FlasherBitStreamer.exe -aPROGRAM Altera_5M570ZN_DevBoard.jam

SEGGER Flasher BitStreamer V1.20 compiled Sep 24 2025 12:58:12
Copyright (c) 2024-2025 SEGGER Microcontroller GmbH    www.segger.com

Diagnostics:
  Altera_5M570ZN_DevBoard.jam:1141: warning: 'NEXT I' is not structurally paired with a 'FOR
 I' in procedure 'L108'
  Altera_5M570ZN_DevBoard.jam:1069: warning: FOR-NEXT loops that are not well-structured
 result in suboptimal code

Streaming complete: 0 errors, 2 warnings.

C:> _
```

These warnings can be safely ignored or suppressed using the command-line option **--no-warnings**.

Chapter 5

Command-line options

5.1 Introduction

BitStreamer supports a number of options to control its operation. These are divided into groups and described in the following sections.

However, in the interest of brevity, the full range of option formats accepted by BitStreamer are not described in full in **Syntax** sections — the description below serves as a reference for additional forms that options may take.

5.1.1 Accepted additional syntax for options

An option described with the syntax:

- **`--some-option`** *`value`*

will also be accepted by BitStreamer in the following forms:

- **`--some-option:`***`value`*
- **`--some-option=`***`value`*

That is, an option and its value can be specified in an all-in-one fashion with the option text and value separated by either a colon or equals character.

In the same way, GNU-style underscores can be used in option names rather than the hyphen, so the following forms are also accepted:

- **`--some_option`** *`value`*
- **`--some_option:`***`value`*
- **`--some_option=`***`value`*

5.2 Control options

5.2.1 --silent

Summary

Do not show output.

Syntax

```
--silent
-q
```

Description

This option inhibits all BitStreamer status messages; only diagnostic messages are shown.

See also

--verbose on page 40

5.2.2 --verbose

Summary

Set higher verbosity.

Syntax

```
--verbose
-v
```

Description

This option instructs BitStreamer to print more detailed information about the preparation and programming processes.

See also

--silent on page 39

5.2.3 --freq

Summary

Set default interface frequency.

Syntax

`--freq` *speed*
`-F` *speed*

Description

This option sets the default frequency of the JTAG interface to *speed* Hz.

STAPL or SVF files containing **`FREQUENCY`** statements set the interface frequency explicitly when run. For files that contain no such statement, the default frequency is used.

The default frequency, if no **`FREQUENCY`** statement is present and no **`--freq`** option is given, is 1 MHz.

5.2.4 --indent-output

Summary

Indent application output by two spaces.

Syntax

`--indent-output`

Description

This option instructs BitStreamer to output all application-generated messages by two spaces when run with the **`--run`** command-line option.

See also

--no-indent-output on page 43

5.2.5 --no-indent-output

Summary

Do not indent application output.

Syntax

`--no-indent-output`

Description

This option instructs BitStreamer to output all application-generated messages verbatim without any indendataion.

See also

--indent-output on page 42

5.3 Deployment options

5.3.1 --deploy

Summary

Deploy to Flasher.

Syntax

`--deploy`

Description

Deploy a successfully built package to the Flasher.

See also

--no-deploy on page 45

5.3.2 --no-deploy

Summary

Do not deploy to Flasher.

Syntax

`--no-deploy`

Description

Do not deploy a successfully built package to the Flasher. This is the default.

See also

--deploy on page 44

5.3.3 --erase

Summary

Erase Flasher content before deployment.

Syntax

```
--erase
```

Description

This option instructs BitStreamer to erase the entire content of the Flasher before deploying a successfully built package. Note that erasure happens only if the package is successfully built.

See also

--no-erase on page 47

5.3.4 --no-erase

Summary

Do not erase Flasher content before deployment.

Syntax

`--no-erase`

Description

This option instructs BitStreamer not to erase the entire content of the Flasher before deploying a successfully built package.

This is the default.

See also

--erase on page 46

5.3.5 --run

Summary

Deploy package to Flasher and run.

Syntax

`--run`

Description

This option instructs BitStreamer to deploy a successfully built package to the Flasher and then run it. There is no need to use both **`--deploy`** and **`--run`** as **`--run`** will always deploy a package.

See also

--no-run on page 49

5.3.6 --no-run

Summary

Do not run package after deployment.

Syntax

`--no-run`

Description

This option instructs BitStreamer not ro run a successfully built package after deployment to the Flasher.

This is the default.

See also

--run on page 48

5.4 SVF options

5.4.1 --compress-bitstream

Summary

Compress bitstreams.

Syntax

```
--compress-bitstream
```

Description

This option instructs BitStreamer to compress bitstreams to save space on the Flasher. Standard Flashers have a 128 MB storage capacity, and the Flasher PRO XL has a 2 GB storage capacity. Compressing bitstreams enables large programming files to fit onto the Flasher’s storage.

Compressing bitstreams is the default.

See also

--no-compress-bitstream on page 51

Notes

Only bitstreams for SVF files are subject to compression; STAPL bitstreams are never compressed.

5.4.2 --no-compress-bitstream

Summary

Do not compress bitstreams.

Syntax

`--no-compress-bitstream`

Description

This option instructs BitStreamer not to compress bitstreams.

See also

--compress-bitstream on page 50

5.5 STAPL options

5.5.1 --action

Summary

Compress bitstreams.

Syntax

--action *name*
-a*name*

Description

This option selects the action *name* as the action to perform in a STAPL file when run.

If no **--action** option is given, BitStreamer chooses the action to perform as follows:

- If there is exactly one **ACTION** statement in the STAPL program, that action is selected.
- If there are multiple **ACTION** statements in the STAPL program, and one of them is named **PROGRAM**, the **PROGRAM** action is selected.
- In all other cases, BitStreamer will issue an error diagnostic and require a selection to be made using the **--action** command-line option.

5.5.2 --extended-stapl

Summary

Select SEGGER Extended STAPL syntax.

Syntax

`--extended-stapl`

Description

This option instructs BitStreamer to use the SEGGER Extended STAPL dialect of the STAPL language. SEGGER Extended STAPL makes programming in STAPL easier by adding structured loop statements and multi-way selection.

In the highly unlikely event that a programming script uses one or more of the reserved identifiers of SEGGER Extended STAPL as a variable or procedure name, JEDEC standard STAPL can be selected using the **`--no-extended-stapl`** option.

SEGGER Extended STAPL is selected as the default dialect.

See also

--no-extended-stapl on page 54

5.5.3 --no-extended-stapl

Summary

Select standard JEDEC STAPL syntax.

Syntax

`--no-extended-stapl`

Description

This option instructs BitStreamer to use the standard JEDEC STAPL dialect of the STAPL language.

See also

--extended-stapl on page 53

5.5.4 --define-symbol

Summary

Define preprocessor symbol.

Syntax

-D*name*
-D*name*=*text*

--define-symbol *name*
--define-symbol *name*=*text*

Description

Define the preprocessor symbol *name* to be replaced by *text*. If *text* is not provided, the symbol is defined with empty replacement text.

5.5.5 --include

Summary

Add directory to include path.

Syntax

-I*dir*
--include *dir*

Description

Add the directory *dir* to the end of the list of paths to search for included files.

5.5.6 --offload

Summary

Set virtualized-array offload threshold.

Syntax

`--offload` *`size`*

Description

Set the threshold at which readonly arrays are virtualized and placed on disk to *`size`* bytes. Virtualized arrays are random access and not compressed.

See also

--no-offload on page 58

5.5.7 --no-offload

Summary

Do not virtualize arrays.

Syntax

`--no-offload`

Description

Do not virtualize any array to disk.

See also

--offload on page 57

5.6 Diagnostic options

5.6.1 --warnings

Summary

Issue warnings.

Syntax

`--warnings`

Description

This option instructs BitStreamer to issue warnings for dubious use or inputs. This is the default.

See also

--no-warnings on page 61

5.6.2 --warnings-are-errors

Summary

Elevate warnings to errors.

Syntax

```
--warnings-are-errors
--fatal-warnings
```

Description

This option elevates all warning diagnostics issued by BitStreamer to errors.

See also

--no-warnings on page 61, *--warnings* on page 59

5.6.3 --no-warnings

Summary

Suppress warnings.

Syntax

`--no-warnings`

Description

This option disables all warning diagnostics issued by BitStreamer.

See also

--*warnings* on page 59

5.6.4 --remarks

Summary

Issue remarks.

Syntax

`--remarks`

Description

This option instructs BitStreamer to issue remarks for potential issues during linking. This is the default.

See also

--no-remarks on page 65

5.6.5 --remarks-are-warnings

Summary

Elevate remarks to warnings.

Syntax

`--remarks-are-warnings`

Description

This option elevates all remark diagnostics issued by BitStreamer to warnings.

See also

--no-remarks on page 65, *--remarks* on page 62, , *--remarks-are-errors* on page 64

5.6.6 --remarks-are-errors

Summary

Elevate remarks to errors.

Syntax

```
--remarks-are-errors
```

Description

This option elevates all remark diagnostics issued by BitStreamer to errors.

See also

--no-remarks on page 65, *--remarks* on page 62, *--remarks-are-warnings* on page 63

5.6.7 --no-remarks

Summary

Suppress remarks.

Syntax

`--no-remarks`

Description

This option disables all remark diagnostics issued by BitStreamer.

See also

--no-remarks on page 65, *--remarks-are-warnings* on page 63

Chapter 6

Indexes

6.1 Subject index

