In-System Flash Programming

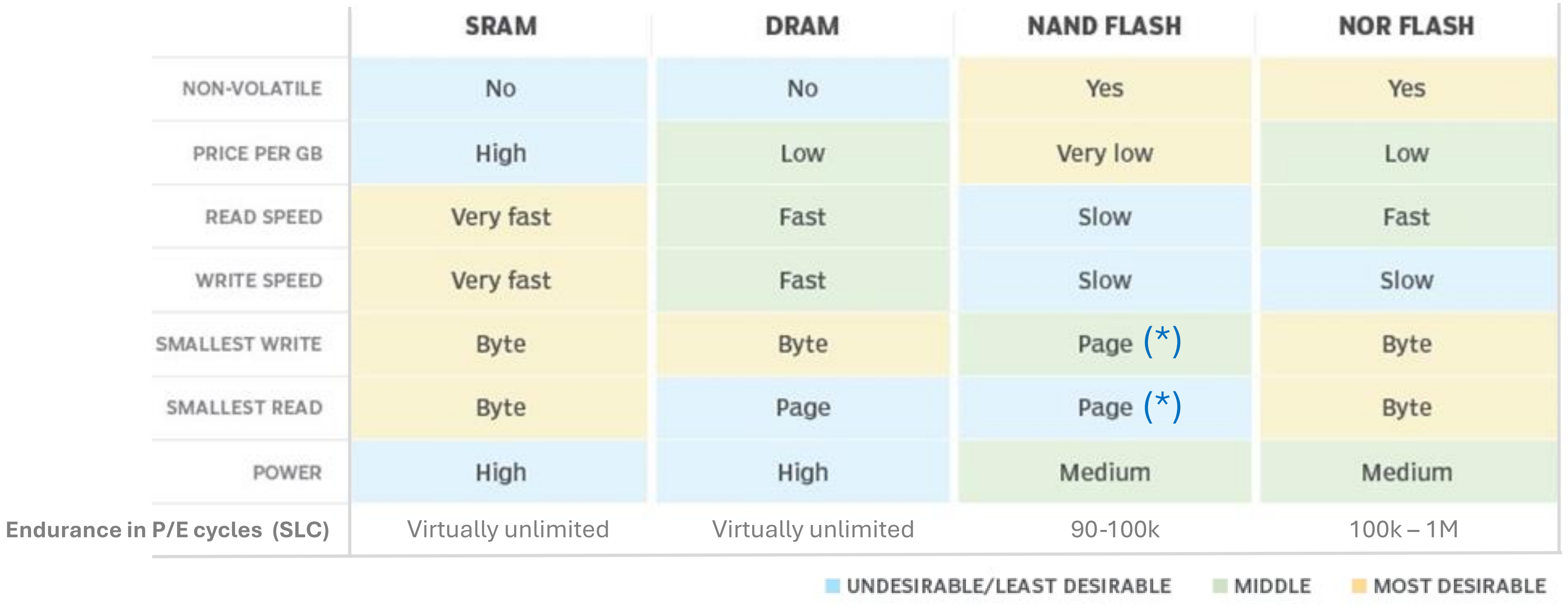
Hoe kan het betrouwbaar, snel en veilig tijdens de ontwikkeling, het onderhoud en in productie.

Gerard Fianen
INDES-Integrated Development Solutions BV

In-System Flash Programming

- Types of Flash memory
- The Flash programming driver
- In-Field & Production programming
- Protecting your IP

Comparing memory types -1



	SRAM	DRAM	NAND FLASH	NOR FLASH
NON-VOLATILE	No	No	Yes	Yes
PRICE PER GB	High	Low	Very low	Low
READ SPEED	Very fast	Fast	Slow	Fast
WRITE SPEED	Very fast	Fast	Slow	Slow
SMALLEST WRITE	Byte	Byte	Page (*)	Byte
SMALLEST READ	Byte	Page	Page (*)	Byte
POWER	High	High	Medium	Medium
Endurance in P/E cycles (SLC)	Virtually unlimited	Virtually unlimited	90-100k	100k – 1M

🟦 UNDESIRABLE/LEAST DESIRABLE 🟩 MIDDLE 🟨 MOST DESIRABLE

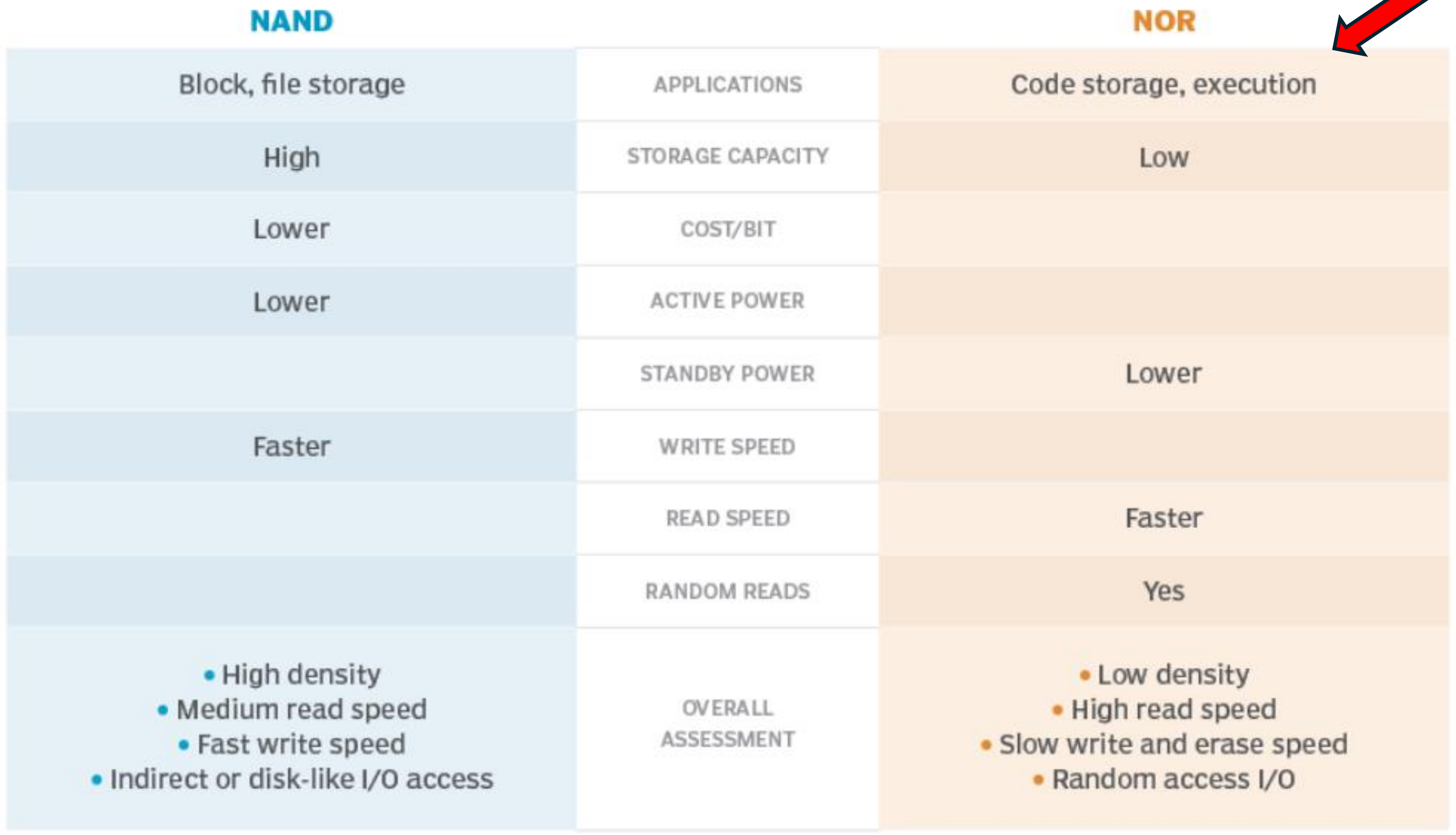
(*) Wear levelling

Rotating blocks of pages around in the NAND Flash, evening out the number of erasures per block

Typically, 64 pages in a block

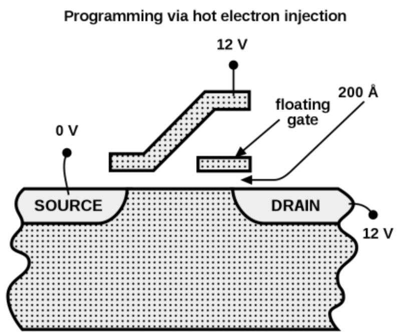
Comparing memory types -2

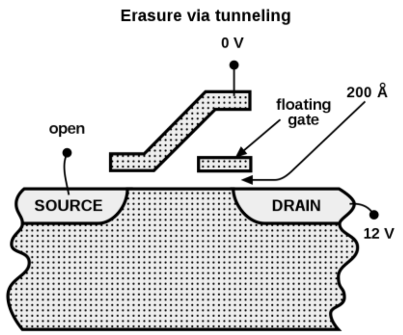
Execute in place !

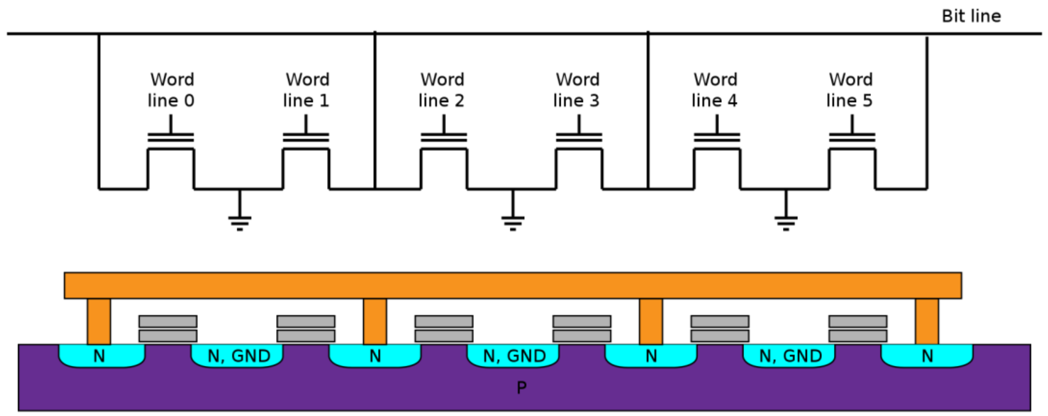


NAND		NOR
Block, file storage	APPLICATIONS	Code storage, execution
High	STORAGE CAPACITY	Low
Lower	COST/BIT	
Lower	ACTIVE POWER	
	STANDBY POWER	Lower
Faster	WRITE SPEED	
	READ SPEED	Faster
	RANDOM READS	Yes
• High density • Medium read speed • Fast write speed • Indirect or disk-like I/O access	OVERALL ASSESSMENT	• Low density • High read speed • Slow write and erase speed • Random access I/O

NOR Flash memory



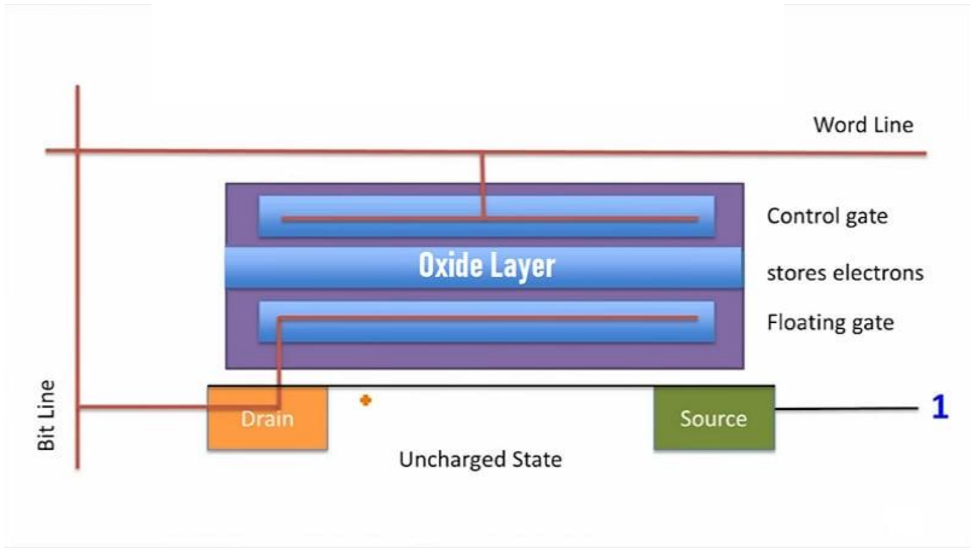


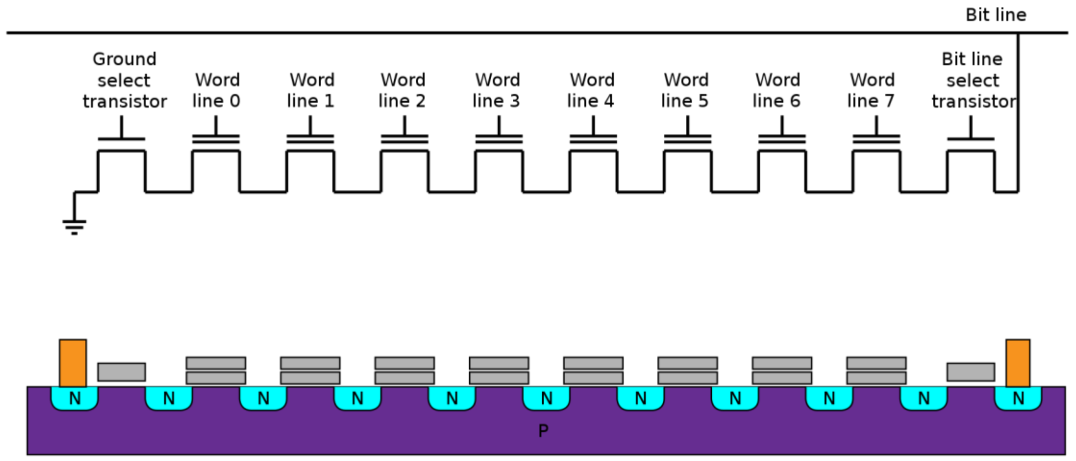


To erase a NOR flash cell (resetting it to the "1" state), a large voltage of the opposite polarity is applied between the CG and source terminal, pulling the electrons off the FG through quantum tunneling.

NOR flash memory chips are divided into erase segments (often called blocks or sectors). The erase operation can be performed only on a block-wise basis; all the cells in an erase segment must be erased together. Programming of NOR cells, however, generally can be performed one byte or word at a time.

NAND Flash memory



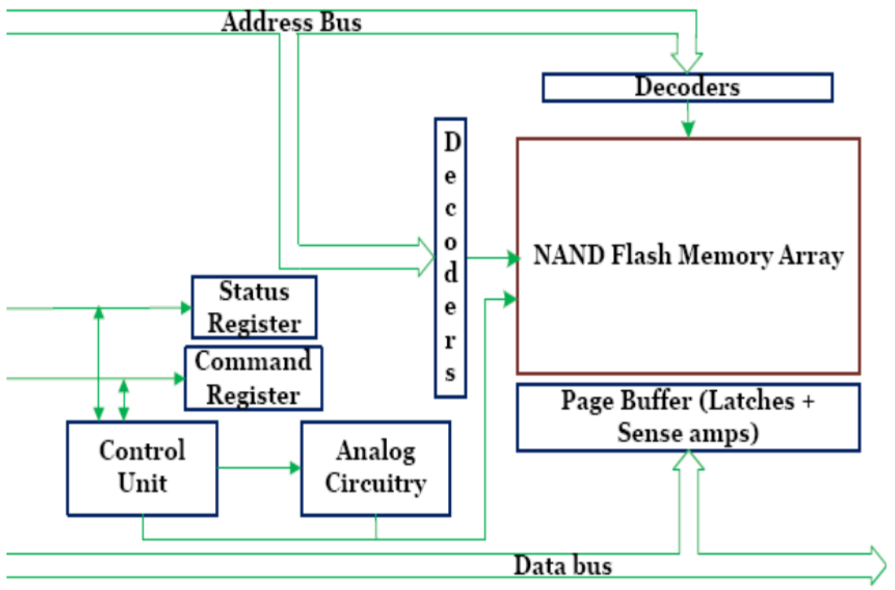


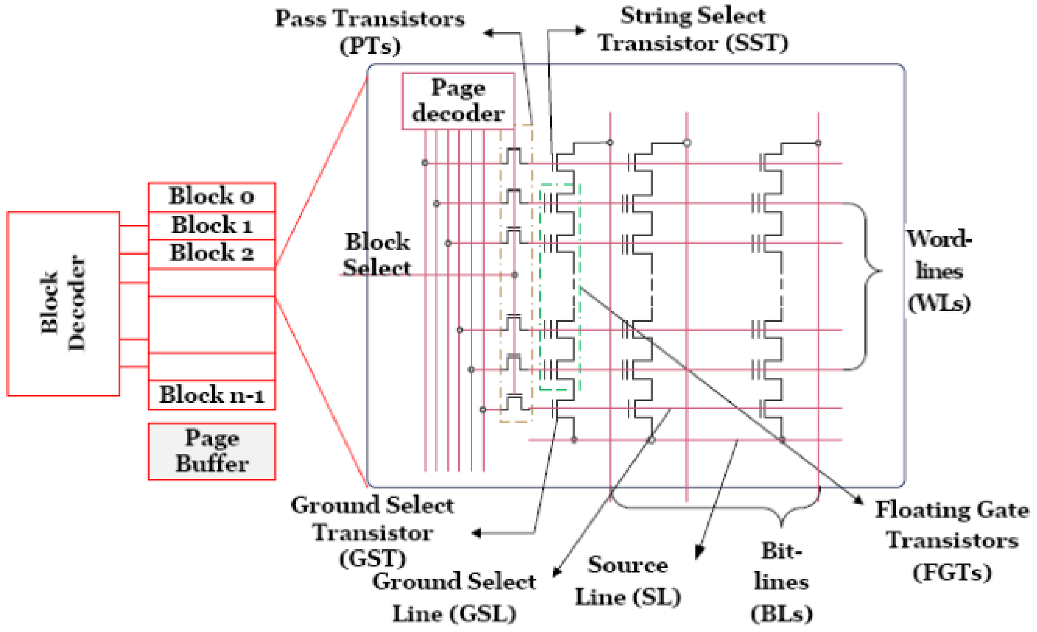
NAND Flash memory has millions of charge trap memory cells. Each Charge Trap flash memory Cell (CTF) can either trap electrons or release them. Hence each cell could represent only one bit either 0 or 1 in the past.

CTF has three functional parts known as Gate, Channel, and Charge Trap. All of these are separated using a dielectric material. The Charge Trap part can store or release the stored electrons in it. If there are no electrons trapped then the cell represents 1, otherwise 0.

Note: Some more modern CFT’s can store more values than the previous ones. This is achieved by storing different charge levels in the Charge Trap part. For example, for representing 8 different values we need 8 different charges or voltage.

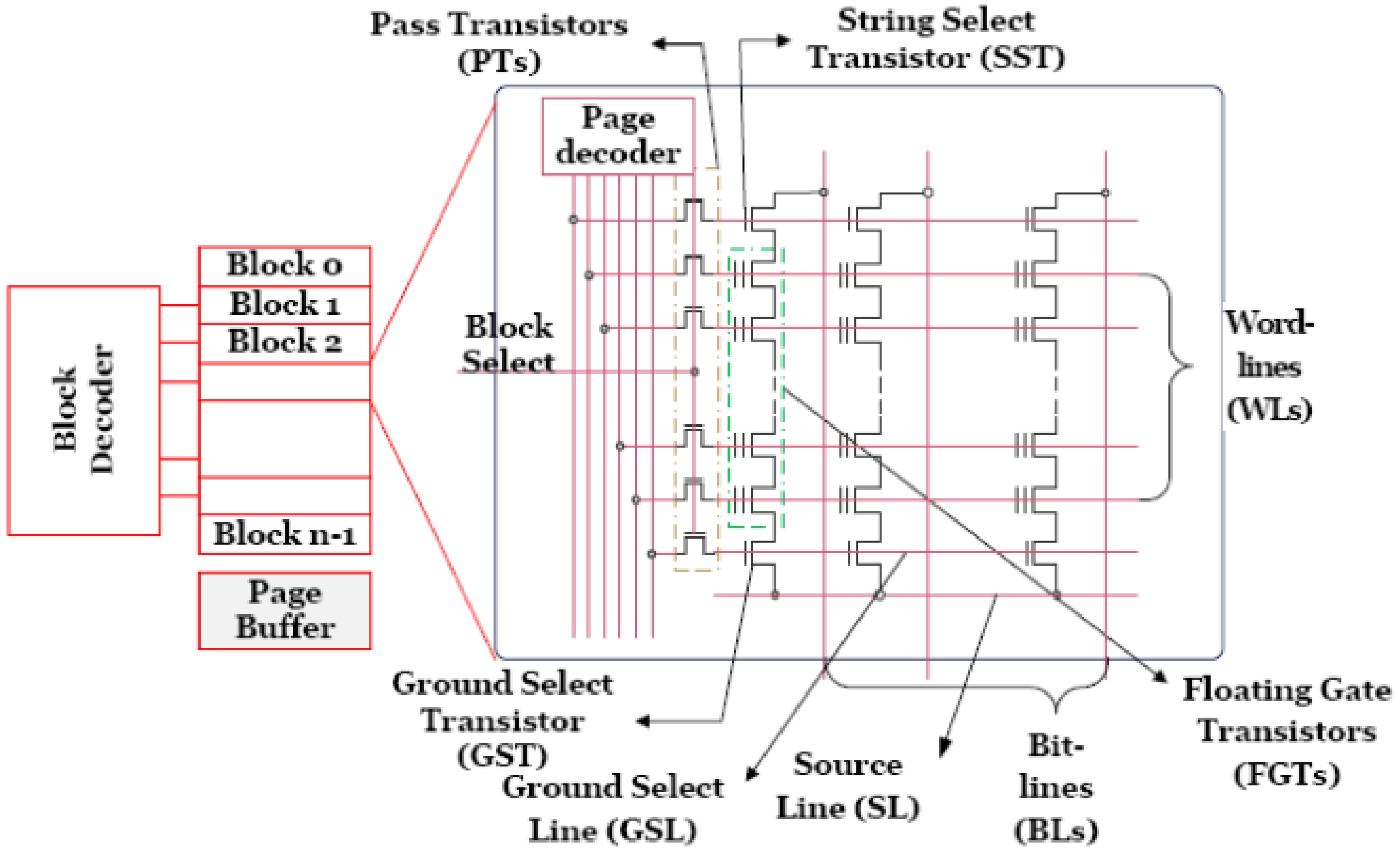
NAND Flash memory block diagram



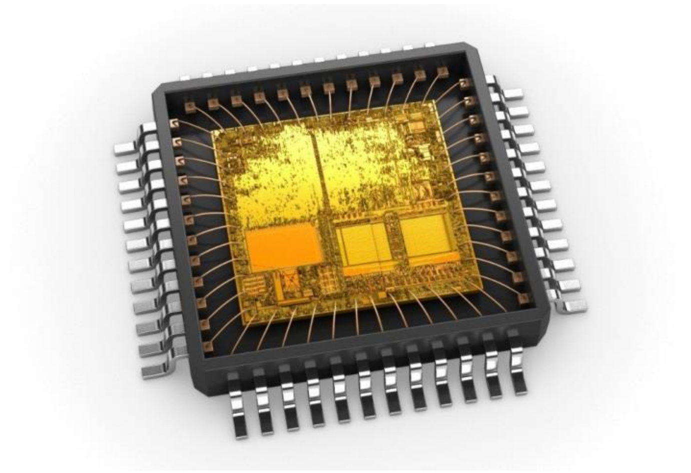


The NAND flash memory array is partitioned into blocks that are, in turn sub-divided into pages. A page is the smallest granularity of data that can be addressed by the external controller. A set of FGTs connected in series is referred to as a string. The number of FGTs in a string is equal to the number of pages in a block. Each column corresponds to a string while each row corresponds to a page. Word lines select a page of memory to perform read or program operation. These operations first involve selecting a block using a block decoder following which one of the rows in a block is selected using a page decoder.

NAND Flash memory block diagram



Scenarios to improve programming time



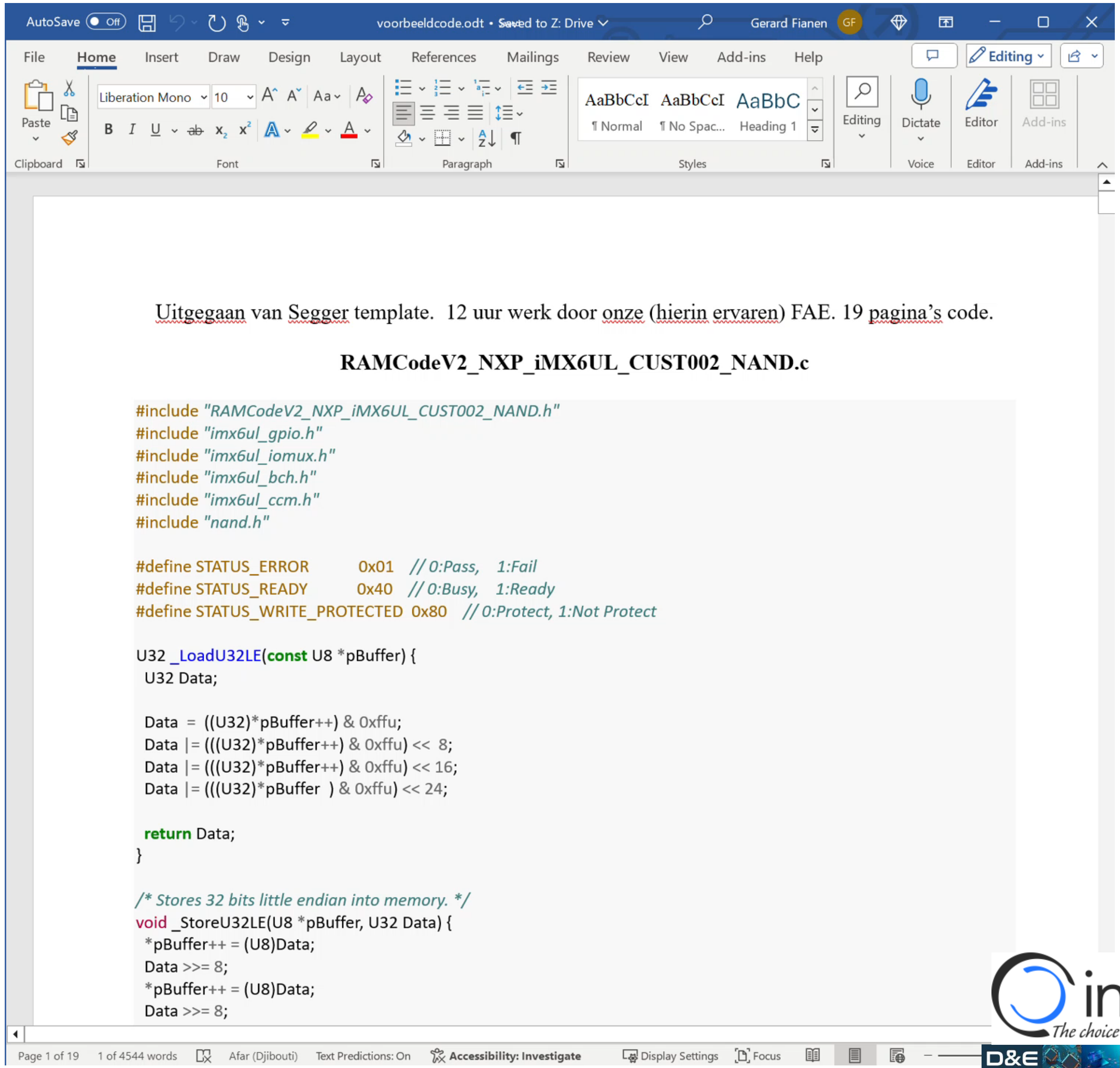
- Download image to RAM --> RAM to Flash (If needed page-by-page)
- Page flipping :
  Download next page to RAM while programing one page
- Programmer can instruct the target to increase MCU clock frequency

Flash Driver ?

- NOT needed if the Flash is in the MCU and the Flasher already knows the specific MCU

- REQUIRED for External NAND flash (or complex targets such as multi-MCU)

- Our FAE has a lot of experience with (also complex) Flash programming drivers


AutoSave Off
voorbeeldcode.odt • Saved to Z: Drive
Gerard Fianen
File Home Insert Draw Design Layout References Mailings Review View Add-ins Help
Editing
Liberation Mono 10
Clipboard Font Paragraph Styles Voice Editor Add-ins
¶ Normal ¶ No Spac... Heading 1
Dictate Editor Add-ins
Page 1 of 19 1 of 4544 words Afar (Djibouti) Text Predictions: On Accessibility: Investigate Display Settings Focus

Uitgegaan van Segger template. 12 uur werk door onze (hierin ervaren) FAE. 19 pagina's code.

RAMCodeV2_NXP_iMX6UL_CUST002_NAND.c

```
#include "RAMCodeV2_NXP_iMX6UL_CUST002_NAND.h"
#include "imx6ul_gpio.h"
#include "imx6ul_iomux.h"
#include "imx6ul_bch.h"
#include "imx6ul_ccm.h"
#include "nand.h"

#define STATUS_ERROR            0x01   // 0:Pass,    1:Fail
#define STATUS_READY            0x40   // 0:Busy,    1:Ready
#define STATUS_WRITE_PROTECTED  0x80   // 0:Protect, 1:Not Protect

U32 _LoadU32LE(const U8 *pBuffer) {
  U32 Data;

  Data  = ((U32)*pBuffer++) & 0xffu;
  Data |= (((U32)*pBuffer++) & 0xffu) <<  8;
  Data |= (((U32)*pBuffer++) & 0xffu) << 16;
  Data |= (((U32)*pBuffer  ) & 0xffu) << 24;

  return Data;
}

/* Stores 32 bits little endian into memory. */
void _StoreU32LE(U8 *pBuffer, U32 Data) {
  *pBuffer++ = (U8)Data;
  Data >>= 8;
  *pBuffer++ = (U8)Data;
  Data >>= 8;
```

indes
The choice of professionals

D&E EVENT Het ontwerpen van innovatieve elektronica
Woensdag 20 maart 2024
1931 Congrescentrum 's-Hertogenbosch

Creating a Flash driver : Pseudo-code -1

```
// low-level byte copy subroutines
// low-level NAND chipselect subroutines
// low-level NAND I/O subroutines
// low-level ECC subroutines

// NAND page I/O subroutines
// mapping logical blocks to physical ones

// main programmer API:

void FLASH_GetDesc() {
  // initialize information about chip dimensions
  // provide information about features supported by driver
}

// hardware initialization
int FLASH_Prepare() {
  // initialize pins (pin select, drive strength, pull-up/down, direction)
  // initialize ECC engine
  // load DBBT – discovered bad block table
}
```

Creating a Flash driver : Pseudo-code -2

```
// read a number of pages into RAM, skipping bad blocks
void FLASH_Read() {
  // map logical blocks to physical blocks (uses DBBT)
  // command the NAND chip to read data, copying each page to RAM
  // run the data through the ECC engine
  // check the bad block marker
  // arrange the data according to the expected layout (user data + meta data)
}

// program a number of pages with data from RAM, skipping bad blocks
void FLASH_Program() {
  // map logical blocks to physical blocks (uses DBBT)
  // arrange the data according to the expected layout
  // run the data through the ECC engine
  // write each page to the NAND and give the command to program
}

// erase the whole chip, or a number of blocks
void FLASH_Erase() {
  // map logical blocks to physical blocks (uses DBBT)
  // command the NAND to erase each block, mark bad blocks as
needed
}

// de-initialization
void FLASH_Restore() {
  // program the bad block information if it was changed
}
```

Creating a Flash driver : Pseudo-code -3

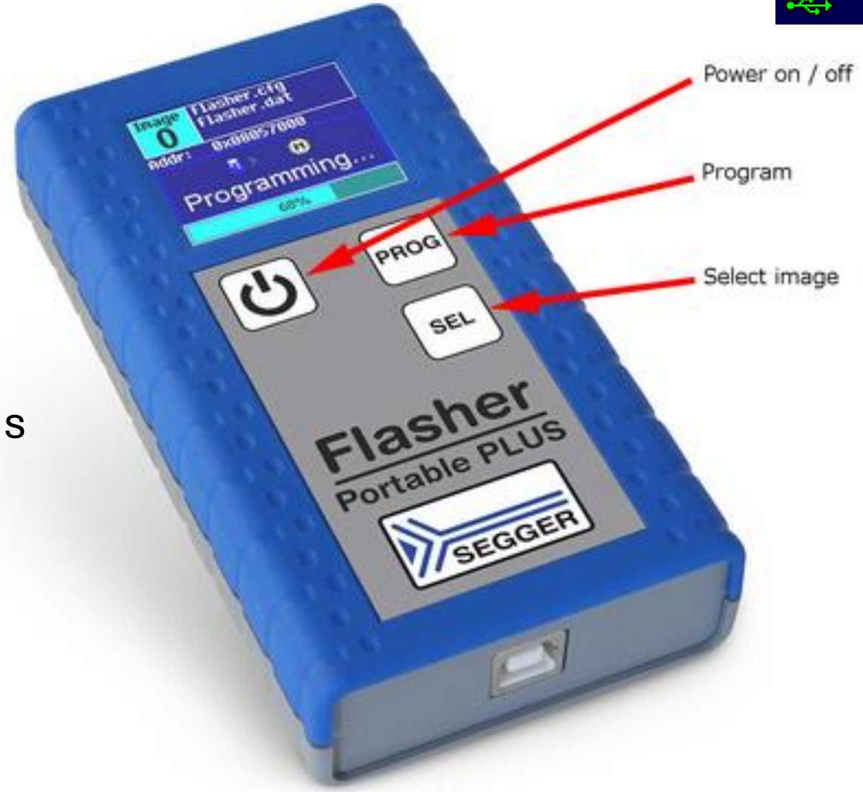
dbbt.c – discovered bad block table

```
/* programs 1 DBBT */
int dbbt_program1()
{
  // program first 2 pages of block with DBBT information
}

/* programs all DBBTs */
int dbbt_program()
{
  // loop over the first 4 blocks
}

// reads bad block marker of a block, returns GOOD, BAD or ERROR
int checkBbm()
{
  // read first page of block
  // extract BBM
}
```

```
// determines whether a block is bad
int isBadBlock()
{
  // consult DBBT in RAM
  // check BBM if necessary, updating DBBT in RAM
}

// marks a block as bad
void markBlockAsBad()
{
  // update DBBT in RAM
}
```

Field Flash Programming

- Li-Ion battery Powered
- Target Voltage 1,2 – 5 Volt
- Target i/f : JTAG / SWD / FINE / SPD
- 128 MB memory for target programs
- Supports multiple MCU families
- Support Authorized Flashing



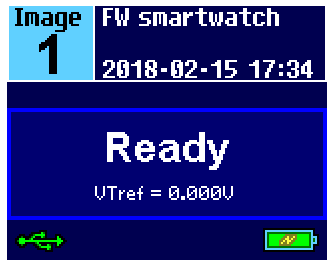


Image 1 FW smartwatch
2018-02-15 17:34
Ready
VTref = 0.000V

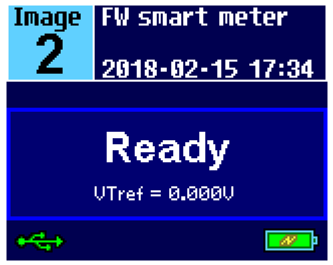
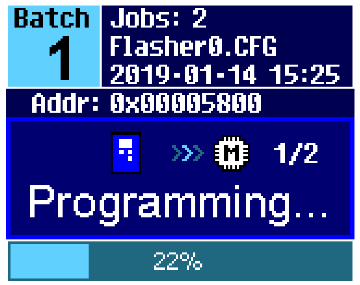
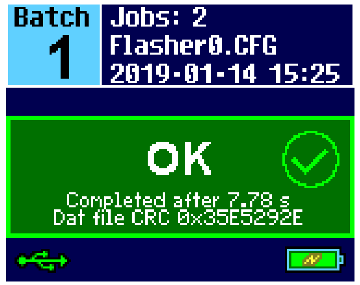
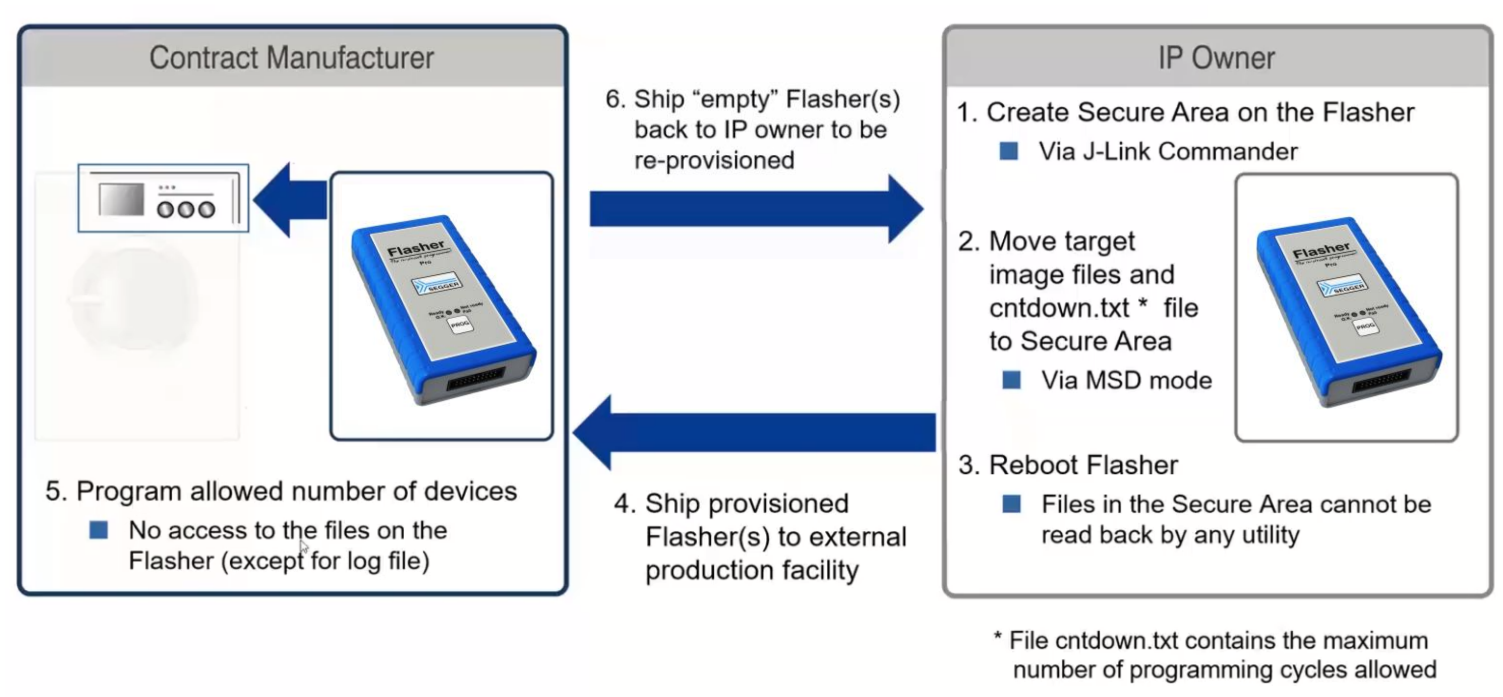


Image 2 FW smart meter
2018-02-15 17:34
Ready
VTref = 0.000V


Batch 1 Jobs: 2
Flasher0.CFG
2019-01-14 15:25
Addr: 0x00005800
1/2
Programming...
22%


Batch 1 Jobs: 2
Flasher0.CFG
2019-01-14 15:25
OK
Completed after 7.78 s
Dat file CRC 0x35E5292E

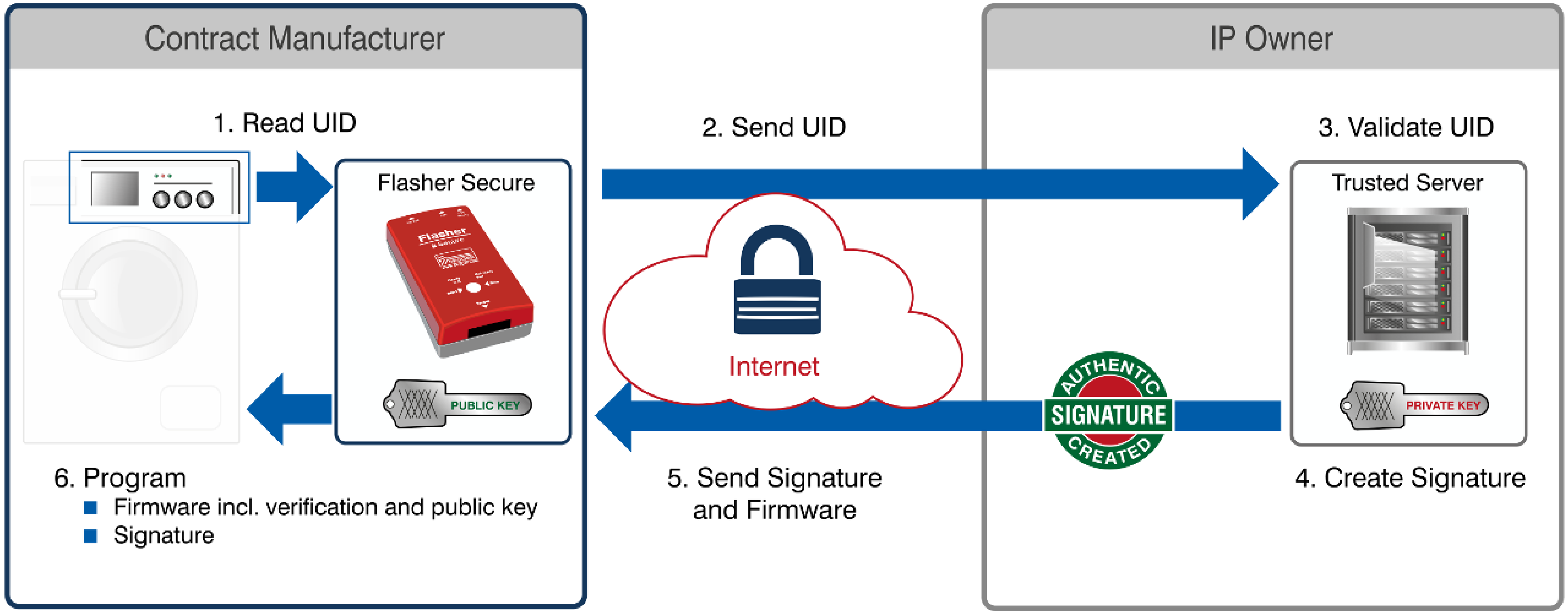
Protecting your IP : Authorized Flashing


Contract Manufacturer
6. Ship "empty" Flasher(s) back to IP owner to be re-provisioned
IP Owner
1. Create Secure Area on the Flasher
Via J-Link Commander
2. Move target image files and cntdown.txt * file to Secure Area
Via MSD mode
3. Reboot Flasher
Files in the Secure Area cannot be read back by any utility
4. Ship provisioned Flasher(s) to external production facility
5. Program allowed number of devices
No access to the files on the Flasher (except for log file)
* File cntdown.txt contains the maximum number of programming cycles allowed

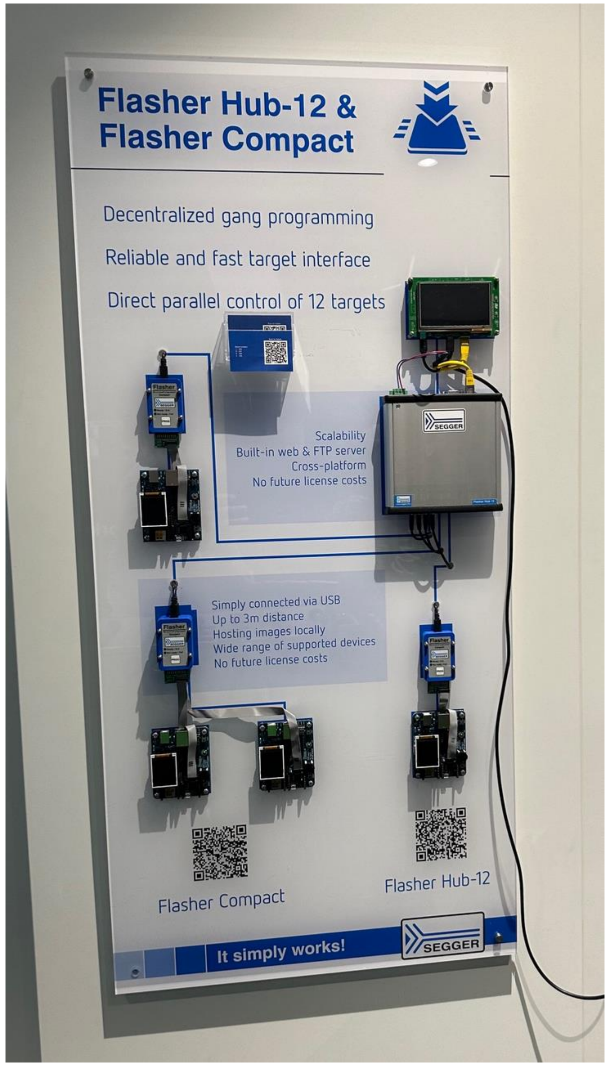
Pre-configure the Flasher with a given setup (Code and number of copies)
- Download the code image in secure area of the Flasher
- Set Maximum number of Flash programming Cycles.

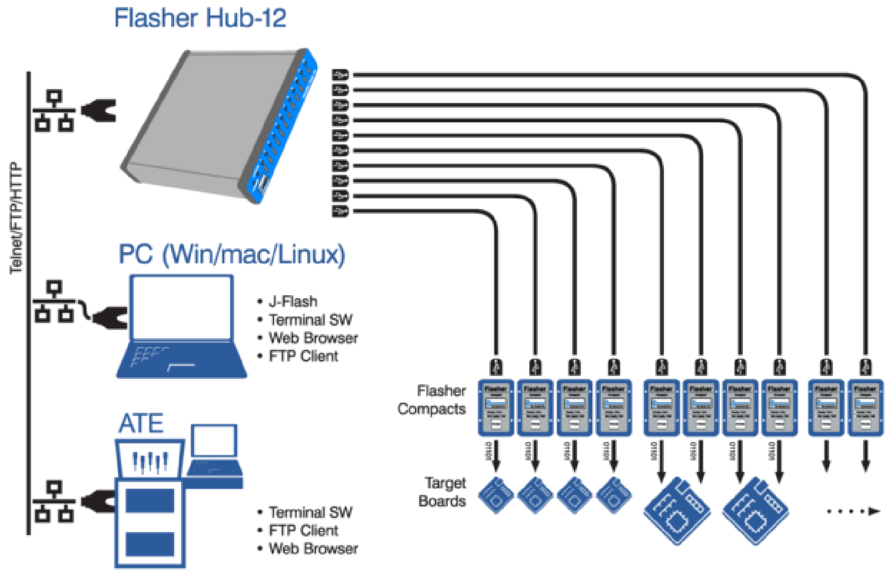
Once the pre-defined number of devices is programmed, the Flasher must be re-programmed to start a new programming cycle.

Protecting your IP : Authenticated Production



Demonstraties op onze stand (stand 11)





INDES –
Integrated Development Solutions BV

indes
The choice of professionals

Cross Compilers, Debuggers, IDE
RTOS, Middleware, Protocol stacks, Security, GUI, EFS
Debug & Trace probes, Emulators
Real-Time Trace, RTOS-Event Trace
Static Analysis, Timing Analysis, Stack Analysis
Unit Test, Code Coverage
(Production) Flash Programming

On-site support
Test-as-a-Service
Verification-as-a-Service

www.indes.com info@indes.com
Tel: 0345 - 545.535

D&E EVENT
Het ontwerpen van innovatieve elektronica
Woensdag 20 maart 2024
1931 Congrescentrum 's-Hertogenbosch