



(51) International Patent Classification:

G06F 8/61 (2018.01)

G06F 9/4401 (2018.01)

G06F 8/654 (2018.01)

B60R 16/02 (2006.01)

(21) International Application Number:

PCT/EP2020/055672

(22) International Filing Date:

04 March 2020 (04.03.2020)

(25) Filing Language:

English

(26) Publication Language:

English

(30) Priority Data:

1903088.1

07 March 2019 (07.03.2019)

GB

(71) Applicant: DELPHI TECHNOLOGIES IP LIMITED

[BB/BB]; Erin Court, Bishop's Court Hill, St. Michael (BB).

(71) Applicant (for GH only): DELPHI FRANCE SAS

[FR/FR]; c/o Delphi Technologies - Campus Saint Christophe Bâtiment Galilée 2 - 10, avenue de l'Entreprise, 95863 Cergy Pontoise CEDEX (FR).

(72) Inventors: **BELLANGER, Gwénolé**; 13b, rue de la Gare, 41000 Villebarou (FR). **LEMORT, Matthieu**; 5, rue de la Poste, 41000 Villebarou (FR).

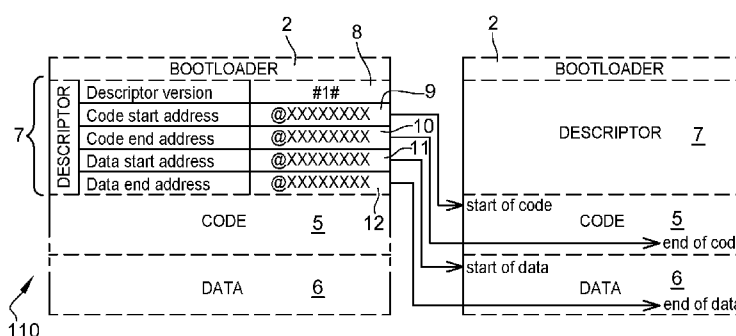
(74) Agent: **DELPHI FRANCE SAS**; Delphi Technologies - Campus Saint Christophe Bâtiment Galilée 2 - 10, avenue de l'Entreprise, 95863 Cergy Pontoise CEDEX (FR).

(81) Designated States (unless otherwise indicated, for every kind of national protection available): AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JO, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, WS, ZA, ZM, ZW.

(84) Designated States (unless otherwise indicated, for every kind of regional protection available): ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ,

(54) Title: AUTOMOTIVE BOOTLOADER

Fig. 3



(57) Abstract: A method of storing or loading an automotive application into an ECU memory, said memory including bootloader software, comprising a) storing, updating or downloading into said memory, an application descriptor, said descriptor comprising metadata for the application, said metadata including 10 memory storage parameters; b) loading application software into said memory according to said memory storage parameters.



TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report (Art. 21(3))*

AUTOMOTIVE BOOTLOADER

5 TECHNICAL FIELD

The invention relates to the division of the memory area (memory mapping) of an automotive memory such as that of an (Engine Control Unit) ECU, which is used to store each part of application software in respect of automotive applications. It also relates to the downloading of such applications and the appropriate downloading of bootloader software; and how the bootloader software is provided and made aware of the split of application code and data so as to save each software part (data and code) of the application(s) at the appropriate location in memory. The invention also relates to a method of boot-loading.

15

BACKGROUND OF THE INVENTION

Currently all bootloader providers include the memory mapping required in the bootloader software. That implies the bootloader software needs to be reworked each time the memory mapping evolves whether during the engine application development or after the entry into service of the vehicle.

Today the engine of a vehicle is not only managed by mechanical and electric elements, but also by an electronic control unit (ECU), the engine control unit. The ECU contains a software whose role is control the engine so as to e.g. to choose the best combustion strategies according to the driving conditions, the whole while limiting the polluting emissions (CO₂, Nox,...).

In order to bring corrective measures to the engine on vehicles in circulation, and taking into account the manufacturing costs of a new engine or the modification of the mechanical characteristics of current one, the manufacturers often decide to modify the ECU, and more particularly the software applications which are stored in the ECU. The manufacturing of new ECU has a high cost. The solution chosen by all the manufacturers is to update only the software application contained in the ECU.

To update the ECU application on a vehicle is more complex than a software update on a computer. One of the strongest constraints comes from the stock room of the software: the ECU and more particularly the hard memory of the
5 microcontroller, the flash memory.

The first stage in the ECU development is the hardware definition (microcontroller and components). So, the quantity of flash memory available for the software storage is defined for all the life cycle of the project. This memory is
10 divided for the parts of the software.

The software application of engine control which is made up of two parts application software which is made up of i) the application code which is instructions carried out by the microcontroller to control the engine and ii)
15 application data: the parameters which make it possible to configure the code according to the equipment installed on the vehicle (manual or automatic gearbox), to the driving conditions.

There is also bootloader software: this is software independent of the application
20 software which makes it possible to write the application code and the application data in the FLASH memory. but also which checks the integrity of the application software before starting the execution of this. Consequently, bootloader software needs to know the memory partitioning in order to carry out these operations. The bootloader is the first software executed by the microcontroller.

25 A software (bootloader or application) is stored in FLASH memory. To be able to run a software must first be copied from FLASH to the RAM memory. The bootloader contains the FLASH memory drivers. When a new software has to be stored in the FLASH memory, the bootloader erases the previous software on the
30 FLASH and then writes the new software on the erased area.

Most of time, the application software respects the initial memory mapping. When the software implements new functions, it is simply stored in a little bigger memory, but within the previously chosen limits

- 5 The memory mapping is defined at the beginning of project. But sometimes during the development, one of the memory zones becomes too small for the content to be placed there. Thus, the memory needs to be partitioned in a different way, with more space for the code and less for data, or conversely. In this case, if the bootloader tries to download the application, at least one of the two parts of
10 the application will be too large compared to the initial memory mapping. The bootloader and the application become incompatible because they have different memory mapping needs.

It is an objective to provide a system to provide a corrective version of the
15 bootloader which knows the new memory mapping. This one will be able to download the new application but becomes incompatible with the old one. The new application is downloaded from outside the vehicle and stored in the flash memory with the boot loader; the bootloader receives the download of the new application and then manages the FLASH erase/write to store the application

20

SUMMARY OF THE INVENTION

In one aspect is provided In one aspect is provided a method of storing or loading an automotive application into an ECU memory, said memory including bootloader
25 software, comprising

- a) storing, updating or downloading into said memory, an application descriptor, said descriptor comprising metadata for the application, said metadata including memory storage parameters;
b) loading application software into said memory according to said memory storage
30 parameters.

Step b) may comprise loading the application software is performed under bootloader control, said the bootloader reading the metadata in the application

descriptor and loading the data content of said application into memory according to this metadata.

Said data content may comprise application data and or application code

5

Said application may contains application code and/or application data; and said metadata includes memory storage parameters for said application code and/or application data.

10 Said memory storage parameters may include the start and or end address in memory of the application code and/or application data.

The application code and/or application data may be stored into one or more sectors/blocks in memory and the memory storage parameters include the start and end addresses for each block or pointers to said addresses

15

Said meta data may include the number of blocks and/or block identifiers.

Said metadata includes an application version or index number for the application and/or descriptor version.

20

Said descriptor may have fixed memory allocation.

Said metadata may include, for application data and/or application code, the number of blocks of each or in total

25

One or each block of either application code or application data may have verification and /or identification data embedded therein and said metadata includes the address of said verification or identification data.

30

Each block may contain a block descriptor which includes block metadata, where said memory storage parameter include the address of said block descriptor

Said block descriptor contains one or more of the following data: version descriptor index/number; start or end address of block identification data/table; start or end address of block identification verification data/table; start and or end address execution of block.

5

The ECU memory may be a flash memory.

Said application descriptor may be in a fixed location of the memory known to the bootloader.

10

In a further aspect is provided a vehicle ECU memory, said memory including an ECU memory, said memory including bootloader software and automotive application stored thereon, where said memory includes an application descriptor, said descriptor comprising metadata for the application, said metadata including memory storage parameters;

15

Said application may contain application code and/or application data; and said metadata includes memory storage parameters for said application code and/or application data.

20

Said memory storage parameters may include the start and or end address in memory of the application code and/or application data.

The application data and/or application data is to be stored into one or more sectors/blocks in memory and the memory storage parameters include the start and end addresses for each block or pointers to said addresses

25

Said meta data may include the number of blocks and/or block identifiers.

Said metadata may include an application version or index number for the application.

30

Said descriptor may have fixed memory allocation.

Said metadata may include for application data and/or application code, the number of blocks of each or in total

- 5 One or each block of either application code or application data has either verification and /or identification data embedded therein and said metadata includes the address of said verification or identification data.

10 In a further aspect is provided a method of initializing, accessing or running applications by a bootloader including; where said bootloader and application are stored on a memory as described above comprising said bootloader accessing said application descriptor and subsequently accessing said application according to the memory storage parameters in said metadata.

15 Accessing said application descriptor and subsequently accessing said application according to the memory storage parameters in said metadata may comprise writing said application to a RAM from said memory according to the memory storage parameters in said metadata.

20 Said bootloader performing additionally validation and identification checks of said application code or data based on the verification and identification data in said descriptor.

BRIEF DESCRIPTION OF THE DRAWINGS

25

The present invention is now described by way of example with reference to the accompanying drawings in which:

- Figure 1 shows the segmentation of memory for bootloader software and application software in the prior art;
- 30 - Figure 2 shows a known solution to the problem of memory overflow;
- Figure 3 shows an example of the invention;
- Figure 4a and 4b shows examples of memory mapping according to one example;

- Figure 5 shows an example where there is a split of the code and of data according to one example;
- Figure 6 shows an example of the memory according to a further embodiment;
- 5 - Figure 7 shows an example where the code has embedded within code verification and identification tables/data;
- Figure 8 shows a further embodiment.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

10

Prior Art

As mentioned in prior art systems, the memory mapping is defined at the beginning of project. But sometimes during the development, one of the memory zones becomes too small for the contents to be place there. This is illustrated in

15 figure 1 which shows a figure of the segmentation of memory 20 (e.g. ECU flash memory) for bootloader software and application software (the latter divided into code memory and data memory) for various examples 1.1, 1.2, 1.3 and 1.4. On the left-hand side of the figure is shown the unused memory. The memory 10

available for the bootloader is shown by reference numeral 1; bootloader software

20 itself by reference numeral 2. The memory available for application code and application data are shown with reference numerals 3 and 4 respectively. Application code and application data itself are shown in examples 1.1, 1.2, 1.3, 1.4

by reference numerals 5 and 6 respectively. In example 1.4 there is not enough memory storage for application data even though there is more than enough for

25 application code.

Figure 2 shows a known solution to the problem of memory overflow. On the left hand side (fig 2a) the figure illustrate the memory with the problem of example 1.4 of figure 1. In figures 2b and 2c the solution is provided by different versions

30 of the bootloader software which is is downloaded in ordert to re-apportion the memory. Reference numerals are as in figure 1.

A problem with this is that customers often want to work with other suppliers of bootloaders. Some suppliers propose tools to configure memory mapping before generating a bootloader. In all the cases, this implies downloading a new
5 bootloader in the calculator before setting up the new application.

These problems of memory reorganization are very present during the phases of the application development. Nothing is definitively fixed, and the memory organization can be changed constantly. In serial mode, it may also be necessary
10 to update the application software in order to correct some problems appeared after the first marketing. The most outstanding example of ECU massive reprogramming is the recall organized by the Volkswagen group after the “Diesel Gate”. These changes of software on ECU are necessary because they are extremely less expensive than the manufacturing of a new ECU.

15 In order to test the various phases of the application software, the bootloader must be functional at the beginning of the ECU development. This constraint implies that the ECU memory mapping is known by the bootloader from the start of the project so that this one can download on the ECU either the code, the data, or a
20 new bootloader. This memory mapping requires to know at a very early stage know the functional needs for the application.

The development of the application can take several years. During this period, the engineers will have to make various choices of use memory induced by customer
25 requests and constraints.

The download of the software on the ECU will be done by the means of the bootloader which is developed almost in parallel of the ECU. As the bootloader is a rather small software compared to the application of engine control, it will be
30 very quickly fixed. The application of engine control will be it only well later, we thus have a desynchronization of the development cycles.

During serial life, customers ask us to ensure a possible software update during 10 years. Moreover, it can happen that certain customers do not allow us to change the boot as of the start of production. Consequently, it is necessary to find solutions to bring flexibility to bootloaders.

5

The main challenge is not the very rigid memory mapping but rather in the precocity of this static memory mapping compared to the life cycle of the project. The setting in production of the boot intervenes well before the end of the development of the application.

10

Invention

In aspects of the invention the memory mapping from the bootloader software is externalized. This is done by including a new (e.g. application) area of memory which contains pointers to all the start and end addresses for each memory area in respect of one or more applications. This new application area is hereinafter referred to as an “application descriptor”; it has a similar function as an address book. This application descriptor allows the same bootloader software to be kept even if the memory mapping is to be updated.

20

This application descriptor provides the ability to store each portion of an application independently. Preferably the application descriptor include other addresses (pointers) needed by the bootloader to check and identify each application part. This allows e.g. identification and checking tables or data to be included in the concerned application area (code and or data) .

25

Preferably the application descriptor makes the bootloader completely independent of the memory mapping and especially of the number of application areas. The application descriptor may contain only the number of application areas and for each area, its start and end addresses, and optionally an area descriptor address. So, each value of an area needed by the bootloader can be read thanks to the area descriptor content.

30

Detailed Description of the Invention

Example 1

- 5 So in other words in one aspect application memory mapping is made dynamic and evolutionary, and thus independent of the boot by the use of a further (3rd) part in the software application: which is referred to a the descriptor of application. This descriptor may have a fixed memory allocation. It is preferably thus the only memory zone which is always at the same place thereafter.

10

- Figure 3 shows an example of the invention , which shows e.g. a (ECU) memory 110 (changed this in figure also form 10) which includes a descriptor portion 7 which contains meta data for one or more or each application, such metadata including predominantly start and finish addresses for block of application code and/or data.. Other reference numerals are as hereinbefore. The main functional content of the application descriptor is the localization (pointer) of the start and of the end (addresses in FLASH memory) of the application code and also that of the application data.

- 20 The descriptor memory portion may contain the descriptor ID/version data 8 (in respect of that application), and as mentined will store the code start address 8, code end address 9, data start address 10 and data end address 11. These memory localizations will be called the addresses of start and end of each application part. The memory is shown on the left hand side and is duplicated for explanation purposes on the right hand side; the arrow between the bocks show how the application descriptor memories are mapped by pointing ot the address start and finish for code and data from the descriptor.

- Each time a complete application needs to be downloaded on the ECU memory, 30 the bootloader knows its own place in memory and will download all the application on the remaining memory. The bootloader does not worry about cutting between the code and the data. Only the address of the application descriptor must remain fixed.

If only the application data must be changed (thus without change of memory mapping), the bootloader comes to read the current application descriptor contained in the flash memory of the ECU where the bootloader is also stored)) and more particularly the addresses of start and end of the data application. Then
5 it can place in the available flash memory the new version of the data. The download of a new code (without data) is held same manner exactly. It is to be reminded that software (bootloader and application) are stored in the FLASH memory. It is important to keep in mind that the application descriptor is included in the application (descriptor + code + data).

10

Whatever the memory mapping between the code and data, the bootloader does not need an update to succeed in interpreting the content of the application descriptor. It is able with the descriptor to know where to place each part of the application. When new software (application data and/or code) is needed a new
15 descriptor is downloaded and stored into memory. When the either of the two parts (code/data) of the application needs to evolve, the bootloader does not need an update, it downloads only one complete application with a new descriptor which contains the modification of memory mapping.

20 In one example a simple software tool may reconfigure the data e.g. addresses of the descriptor, when a new application or version thereof is downloaded. Ideally, in a perfect world, when the application software is built, the application descriptor memory area is reserved and empty. Just after this build process a tool may automatically fill the application descriptor fields with addresses. The goal of
25 this tool would also be to fill some block descriptor fields.

If the bootloader and the application need to share a new data in a new field on a descriptor, the bootloader and the tool configuration have to be updated (increment of the application descriptor version) as will be described hereinafter. If
30 only the application needs a new data in a new field on a descriptor, the bootloader and the tool stay available until this added data does not modify the previous descriptor format (the increment is not needed).

The two examples of memory mapping are shown in the figure 4a and 4b respectively which describes how two applications can be downloaded thanks to the same bootloader. These show two different example of appropriate memory mapping using the descriptor. Again reference numerals are as hereinabove. It is possible to pass from the one to the other as many times as necessary. In the examples the descriptor point to the same code start address, and same data end address, but the end code address and start code data addresses are different to allow different partitioning of the memory.

10 Refined Embodiment

In some cases, the two parts of the application may not be in a contiguous memory portion/segment but may be partitioned in to two or more (several) blocks.

15 The flash memory of a microcontroller is typically pre-cut in indivisible sectors of different sizes. For this reason, each part of the application may need needs several sectors of flash memory. So, the optimal occupation of the flash sectors can require a separation of each of the two parts of the application in various blocks, each one having to be made up of contiguous sectors of flash memory.

This split of code and data makes more complex the memory mapping but has an unquestionable advantage: the time of the download. If cutting by blocks also corresponds to a functional cutting, during an update of only one function of the application, there does not need to download all the application but only the block of code and/or data which contains the functional update. As the download of the application in the ECU is a time-consuming step, this split enables us to decrease times of download of the code or data.

30 Figure 5 shows an example where there is a split of the code and of data. On the left-hand side is the segment/block indicator or number. There are ten flash sectors S1 to S10. S1 and S2 are occupied by the bootloader 2, S3 by the

descriptor 7. The code 5 is split into two portions (S4 and S5) and (S9 and S10).
The data 6 occupies a single portion (covering contiguous sectors S6 to S8)

Here the descriptor to contains additional information such as the addresses of
5 start and end of each block.

Preferably the greatest flexibility is provided by not, but by leaving this choice of
the precise number of segments to the application. Thus the number of segments
and blocks of memory is stored in the descriptor. and implicitly the number of
10 segments for each block (with the start and end addresses of each block

Figure 6 shows an example of the memory according to a further embodiment
where the descriptor 7 refers to two code blocks and three data blocks. The
figure on the left-hand side again represent the memory which details the
15 information in the descriptor. Again the descriptor includes a version number or
ID. In addition, there is a field which shows the number of code blocks and a field
which shows the number of data blocks. For each code and data block, the
descriptor has two fields which shown the start and end addresses therefor.

20 The figure on the right shows the same memory with the corresponding code and
data blocks in greater detail. The arrows again show the mapping.

Further Embodiment

25 When the bootloader has finished downloading an application, it sometimes is
required to read some values contained in the code and in data to control or verify
that the download was correctly performed and to make sure that the segment
was not modified or “hacked” before the download. For this reason, the descriptor
may contain additional fields. These may point to addresses in either the code or
30 data which contain verification data/tables or identification data/tables. These
values may be regarded as tables or data for checking. Each segment contains its
own table/data of checking.

Then the bootloader will check that all the parts of the application on the microcontroller are coherent between them thanks to tables of identification. Each table is stored in the block which it identifies.

- 5 Between two versions of application, the identification and checking tables can be moved because of a new memory mapping or of new organization of the content in each block of the application. As these tables must be read by the bootloader, the addresses of those are stored in the descriptor of application.
- 10 Figure 7 shows an example where the code has embedded within code verification 13 and identification tables/data 14, and the data has embedded therein data verification 15 and data identification tables/data 16. On the left had side is shown expanded the contents of the descriptor which as well as the start and end addresses of each block may contain the addresses of the data and code
- 15 identification and verifications tables/data. All these are stored at specific addresses which are mapped from the application descriptor; so the descriptor contains additionally the address of the code and/or data verification and checking tables.

20 General

Each time an update is made (i.e. new application or new application data/code) the application descriptor is amended, and the version / or its identifier of version which is contained in the descriptor is amended/incremented.

25

- In order for the bootloader to understand this new format of descriptor, an appropriate (new) bootloader is required. The backward compatibility of the new bootloader can be kept with all the previous models of the descriptor and thus of the applications concerned. Nevertheless, this concept allows a large reduction of
- 30 the number of bootloaders provided for each developed project.

The increment of the descriptor version is only done when there is a change on the content of the descriptor table (new field added or reorganization of the existing fields). If only fields content are updated, the descriptor version does not evolve.

The descriptor version is a version of the descriptor table architecture, how it is defined. Therefore, we need a new boot when the descriptor table version is updated but we can keep the backward compatibility thanks to this version field.

- 5 Even if the inclusion of the descriptor in the application makes the system more complex than before, the various versions contained in the descriptor are a proof of security and reliability in order to avoid putting together incompatible bootloader and application.

10 Further Considerations

- Besides being different sizes, all the flash sectors of a microcontroller do not have the same properties such as speed of access, speed of execution, secured sector, etc. The increasingly complex and precise applications preferably require some
15 portions of the code or the data to be placed at specific flash sectors.

- As described above, the descriptor is a reference towards the place where information is stored. In the art this reference is very often called a “pointer” because the content of the descriptor references the address. The use of these
20 pointers/descriptors should always be made with caution because the contents of the descriptor have to be initially checked before reading, i.e. to check that the pointed address exists and is in a valid block/sector.

- However, the application descriptor may become too complex and too dependent
25 on the application. Indeed, each time the application adds a new value which can be read by the bootloader, the application descriptor and the bootloader must be amended . The next update of the application descriptor preferably contains the number of blocks, for each block: its addresses of start and end as well as the address of the block descriptor.

- 30 If the data to add in the descriptor have to be read by the bootloader, the increment of the application descriptor is needed. Else if the data are only used by the application, the increment is not needed but the architecture changes. Instead of update the application descriptor and increment its version, we choose to simplify the application descriptor with only the start and end addresses and also the

address of block descriptor. Each block descriptor will start with the data needed by the bootloader, and then the data needed by the application. In this way, even if some data are added in the block descriptor, the bootloader always understand the first part and the data its needs to read.

5

This one would contain the addresses of the various tables which need to be read by the bootloader.

Figure 8 shows a further embodiment. The central figure again shows an ECU
10 memory divided into bootloader, application descriptor, code block (s) and data block(s) which are lumped together to refer to general blocks 18. The table on the right side of the figure expands the contents of the application descriptor. This include an ID or reference of the descriptor version, the number of blocks for the application (in total e.g.) and for each block the start and end addresses as well as ,
15 for each block) the address of the block descriptor in the block itself. It is to be noted that the application descriptor and its contents should be distinguished from the block descriptor.

Each block has a block descriptor. On the left-hand side of the figure is shown the
20 content of the block descriptor for block 1 and block n. In each block descriptor 19 is the following information: block descriptor version. Again, optionally may be provided the addresses of a block verification table/data and /or block identification table/data. In addition is stored the start address of execution of the block

25

Finally, the application descriptor is a part of the application and needs to be downloaded on the ECU As for the other blocks, a checking is necessary at the end of its download.

30 The application descriptor preferably should always remain at the same place. So the application descriptor may be placed preferably in the code 1 block. This block thus has a double specificity because it integrates also the address of

beginning of the application. Except for this code 1 block, all the other blocks do not need more to be differentiated between code and data.

CLAIMS

1. A method of storing or loading an automotive application into an Engine
5 Control Unit (ECU) memory, said memory including bootloader software,
comprising
a) storing, updating or downloading into said memory, an application descriptor,
said descriptor comprising metadata for the application, said metadata including
memory storage parameters;
10 b) loading application software into said memory according to said memory storage
parameters, wherein the application descriptor is not part of the bootloader
software, and wherein the application descriptor is received by the bootloader as
part of the application software itself.
- 15 2. A method as claimed in claim 1 wherein in step b) loading the application
software is performed under bootloader control, said the bootloader reading the
metadata in the application descriptor and loading the data content of said
application into memory according to this metadata.
- 20 3. A method as claimed in claim 2 wherein said data content comprises
application data and or application code
- 25 4. A method as claimed in claims 1 to 3 where said application contains
application code and/or application data; and said metadata includes memory
storage parameters for said application code and/or application data.
- 30 5. A method as claimed in claims 1 to 4 wherein said memory storage
parameters include the start and or end address in memory of the application code
and/or application data.

6. A method as claimed in claim 1 to 5 wherein the application code and/or application data is to be stored into one or more sectors/blocks in memory and the memory storage parameters include the start and end addresses for each block or pointers to said addresses

5

7. A method as claimed in claim 6 wherein said meta data includes the number of blocks and/or block identifiers.

8. A method as claimed in claims 1 to 7 wherein said metadata includes an application version or index number for the application and/or descriptor version.

10

9. A method as claimed in claims 1 to 8 where said descriptor has fixed memory allocation.

10. A method as claimed in claims 6 to 9 wherein said metadata includes for application data and/or application code, the number of blocks of application data and/or application code or the total number of blocks for application data and application code

15

11. A method as claimed in claim 6 to 10 wherein one or each block of either application code or application data has either verification and /or identification data embedded therein and said metadata includes the address of said verification or identification data.

20

12. A method as claimed in claims 6 to 11 where each block contains a block descriptor which includes block metadata, where said memory storage parameter include the address of said block descriptor

25

13. A method as claimed in claim 12 where said block descriptor contains one or more of the following data: version descriptor index/number; start or end address of block identification data/table; start or end address of block identification verification data/table; start and or end address execution of block.

30

14. A method as claimed in claims 1 to 14 where ECU memory is a flash memory.

15. A method as claimed in claims 1 to 15 where said application descriptor is in a fixed location of the memory known to the bootloader.

5

16. A vehicle ECU memory, said memory including including bootloader software and automotive application stored thereon, where said memory includes an application descriptor location, for receiving said application descriptor itself, as part of the application software, said descriptor comprising metadata for the application, said metadata including memory storage parameters;

10

17. A memory as claimed in claim 16 said application contains application code and/or application data; and said metadata includes memory storage parameters for said application code and/or application data.

15

18. A memory as claimed in claim 16 or 17 wherein said memory storage parameters include the start and or end address in memory of the application code and/or application data.

20 19. A memory as claimed in claim 16 to 18 wherein the application data and/or application data is to be stored into one or more sectors/blocks in memory and the memory storage parameters include the start and end addresses for each block or pointers to said addresses

25 20. A memory as claimed in claim 19 wherein said meta data includes the number of blocks and/or block identifiers.

21. A memory as claimed in claim 16 to 20 wherein said metadata includes an application version or index number for the application.

30

22. A memory as claimed in claim 16 to 21 where said descriptor has fixed memory allocation.

23 A memory as claimed in claim 19 to 22 wherein said metadata includes for application data and/or application code, the number of blocks of application data and/or application code or the total number of blocks for application data and application code.

5

24 A memory as claimed in claim 19 to 23 wherein one or each block of either application code or application data has either verification and /or identification data embedded therein and said metadata includes the address of said verification or identification data.

10

25. A method of initializing, accessing or running applications by a bootloader including; where said bootloader and application are stored on a memory as claimed in claims 16 to 24 comprising said bootloader accessing said application descriptor and subsequently accessing said application according to the memory storage parameters in said metadata.

15

26. A method as claimed in claim 25 where accessing said application descriptor and subsequently accessing said application according to the memory storage parameters in said metadata comprises writing said application to a RAM from said memory according to the memory storage parameters in said metadata.

20

27. A method as claimed in claim 25 or 26 said bootloader performing additionally validation and identification checks of said application code or data based on the verification and identification data in said descriptor.

25

Fig. 1

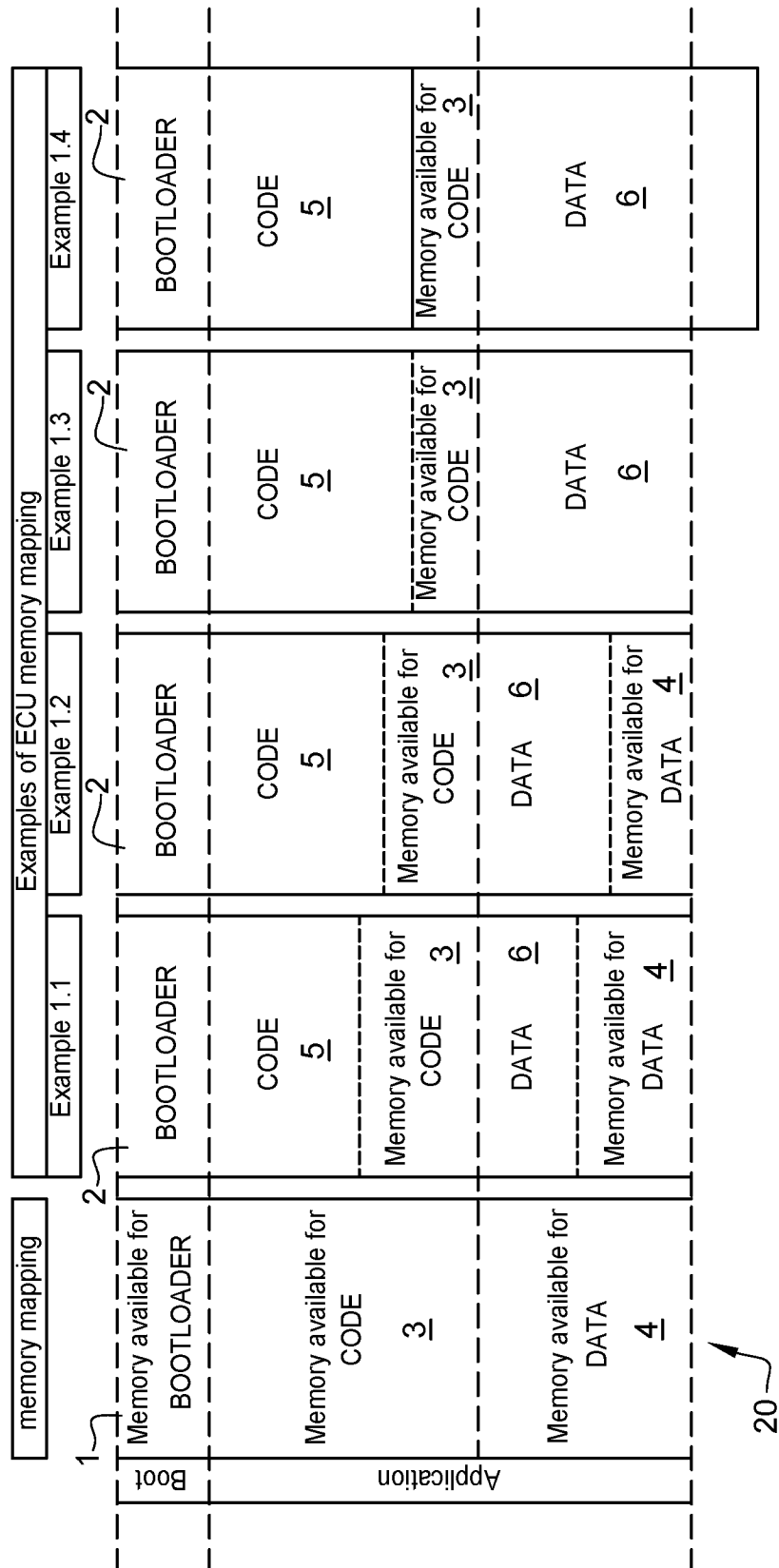


Fig. 2

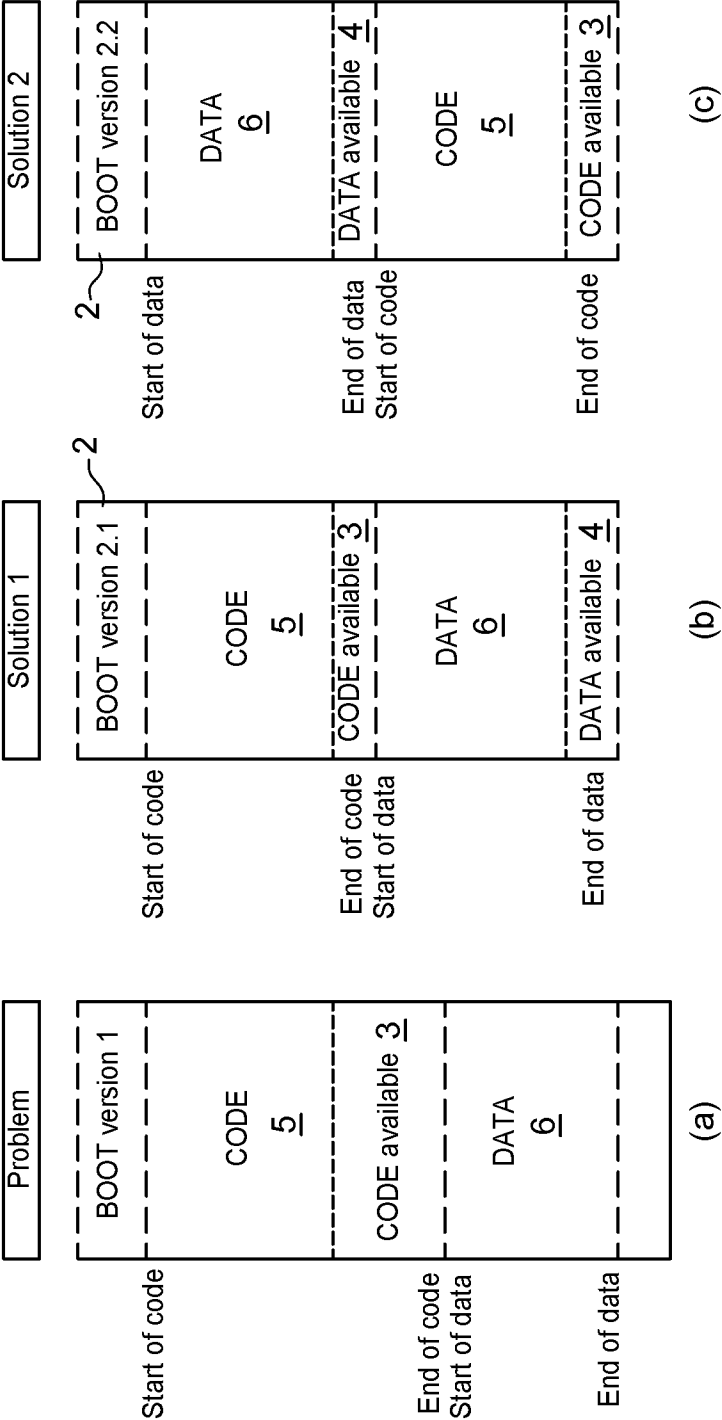


Fig. 3

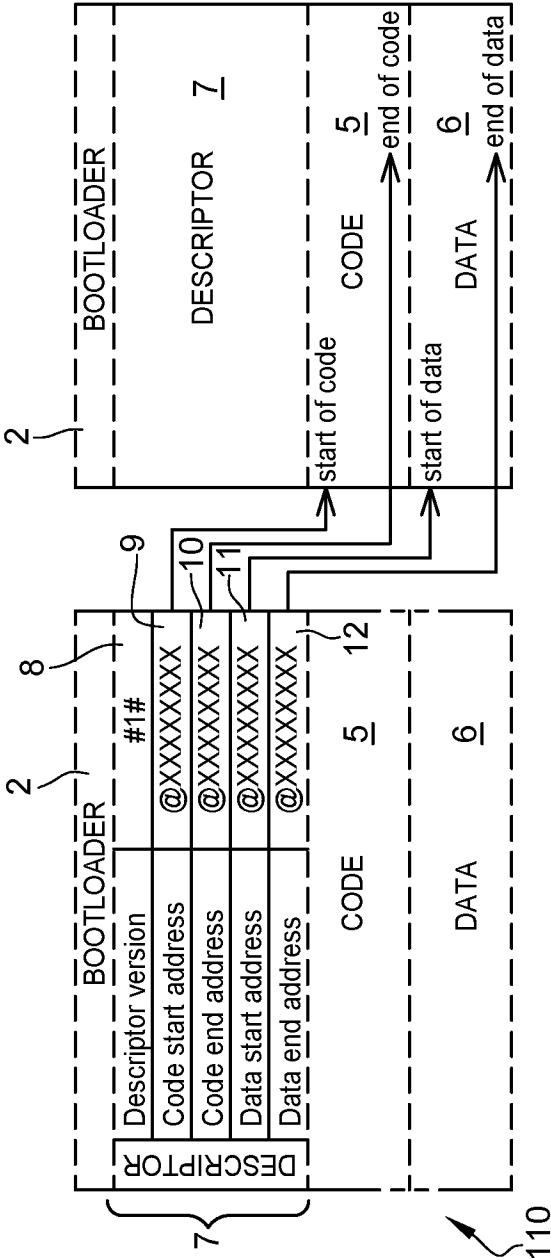


Fig. 4

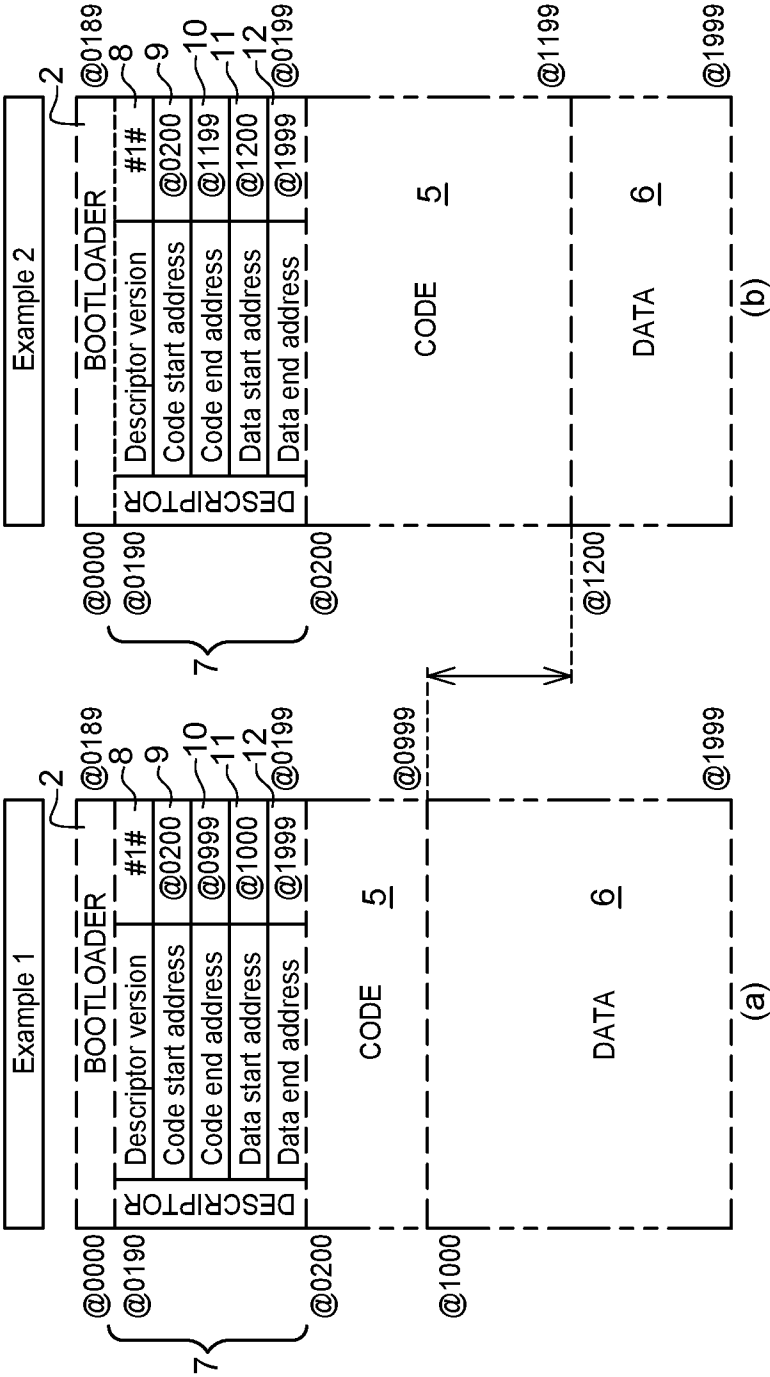


Fig. 5

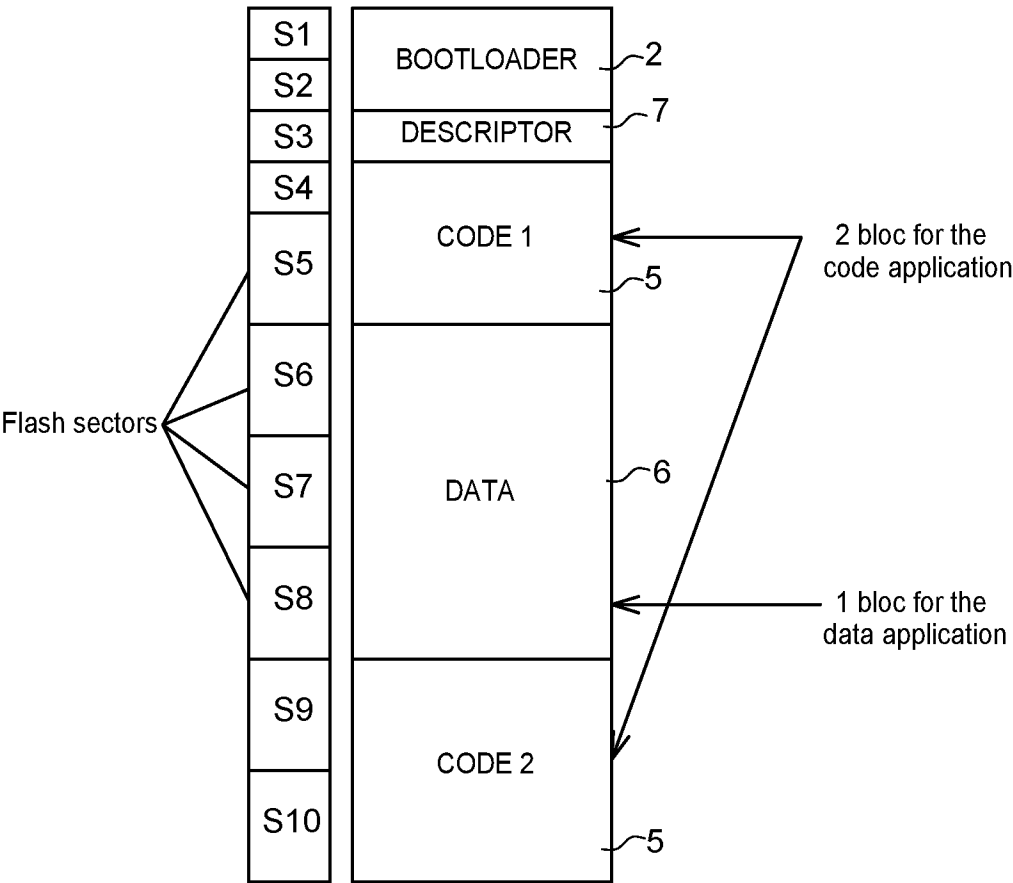


Fig. 6

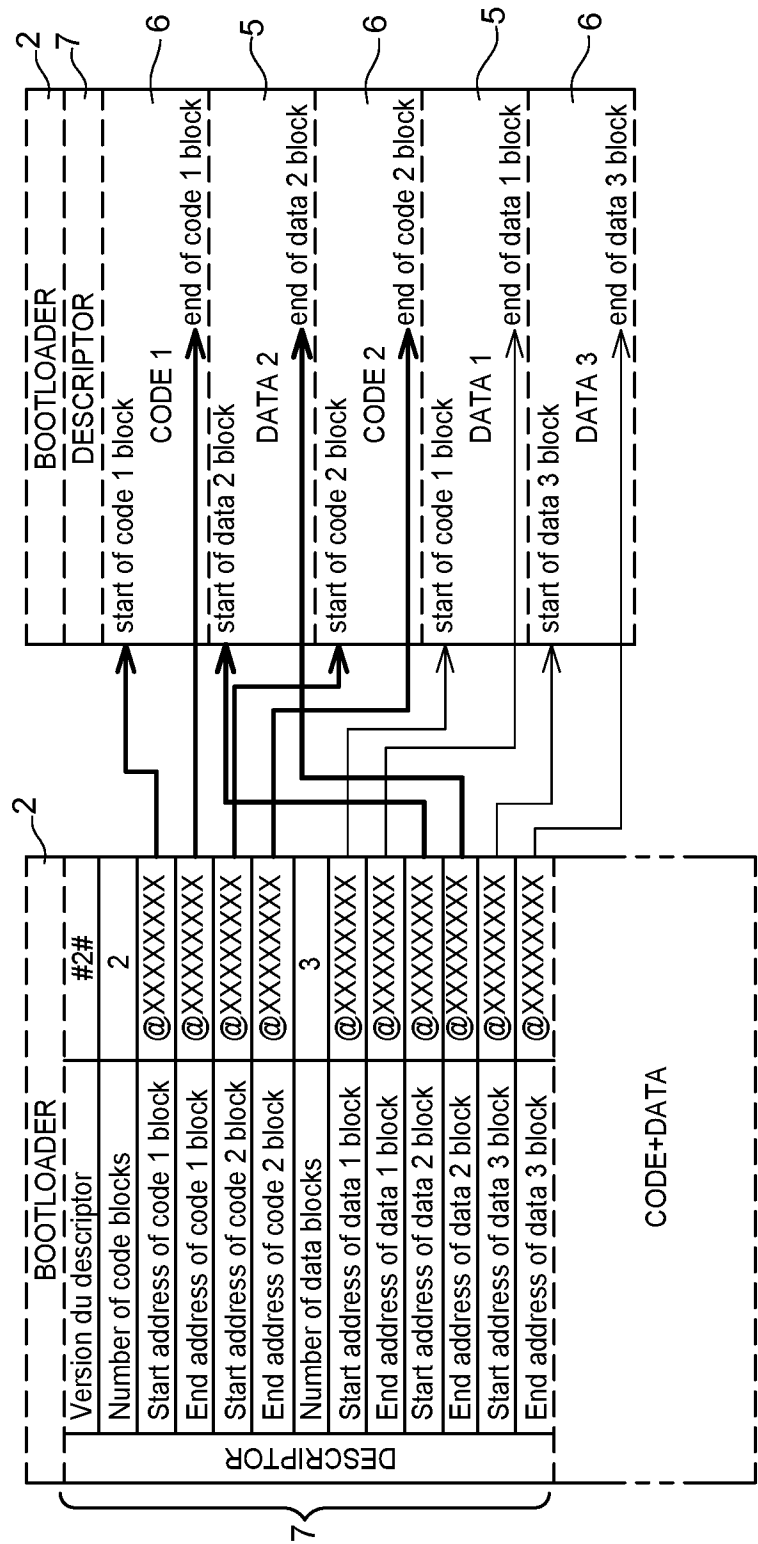


Fig. 7

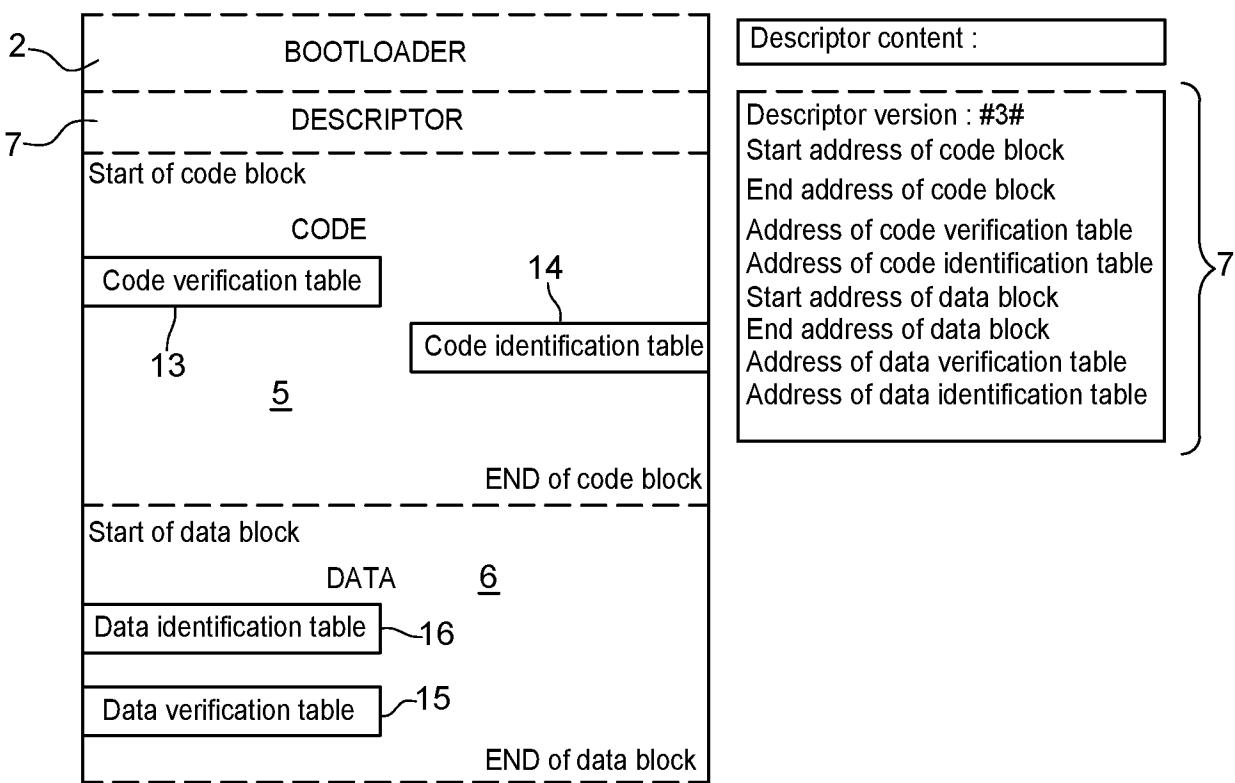
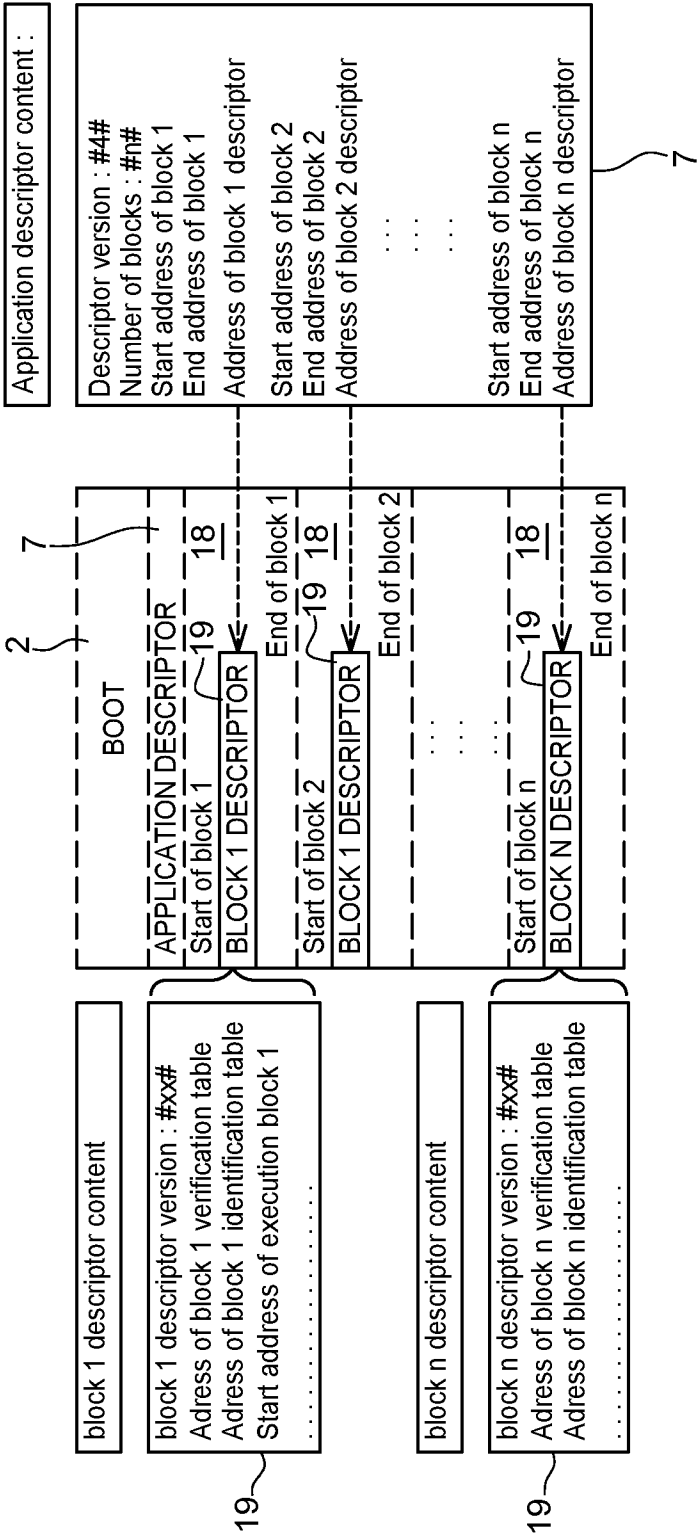


Fig. 8



INTERNATIONAL SEARCH REPORT

International application No
PCT/EP2020/055672

A. CLASSIFICATION OF SUBJECT MATTER

INV. G06F8/61 G06F8/654 G06F9/4401
ADD. B60R16/02

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

G06F B60R

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

EPO-Internal, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2011/208956 A1 (FISLAGE MARKUS [DE]) 25 August 2011 (2011-08-25) the whole document -----	1-27



Further documents are listed in the continuation of Box C.



See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search

5 June 2020

Date of mailing of the international search report

25/06/2020

Name and mailing address of the ISA/

European Patent Office, P.B. 5818 Patentlaan 2
NL - 2280 HV Rijswijk
Tel. (+31-70) 340-2040,
Fax: (+31-70) 340-3016

Authorized officer

Steinmetz, Christof

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No

PCT/EP2020/055672

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2011208956 A1	25-08-2011	AT 532132 T	15-11-2011
		CN 102124447 A	13-07-2011
		DE 102008041360 A1	25-02-2010
		EP 2318920 A1	11-05-2011
		ES 2374798 T3	22-02-2012
		US 2011208956 A1	25-08-2011
		WO 2010020465 A1	25-02-2010
