Internal distribution code:

(A) [-] Publication in OJ
(B) [-] To Chairmen and Members
(C) [-] To Chairmen
(D) [X] No distribution

Datasheet for the decision of 1 February 2024

Case Number:	T 1056/20 - 3.5.06
Application Number:	17173986.5
Publication Number:	3252594
IPC:	G06F7/483, H03M7/24, G06F9/44, G06F17/50
Language of the proceedings:	EN

Title of invention:
SYSTEM AND METHODS FOR GENERATING CODE FROM EXECUTABLE MODELS WITH FLOATING POINT DATA

Applicant:
The MathWorks, Inc.

Headword:
Floating point/Mathworks

Relevant legal provisions:
EPC Art. 56, 111(2)
EPC R. 103(1)(a)
RPBA 2020 Art. 11, 12(6), 13

Keyword:
Inventive step - (no)

Decisions cited:

Catchword:

Europäisches Patentamt
European Patent Office
Office européen des brevets

Beschwerdekammern

Boards of Appeal

Chambres de recours

Boards of Appeal of the
European Patent Office
Richard-Reitzner-Allee 8
85540 Haar
GERMANY
Tel. +49 (0)89 2399-0
Fax +49 (0)89 2399-4465

Case Number: T 1056/20 - 3.5.06

D E C I S I O N
of Technical Board of Appeal 3.5.06
of 1 February 2024

Appellant: (Applicant)	The MathWorks, Inc. 3 Apple Hill Drive Natick, MA 01760-2098 (US)
Representative:	Meissner Bolte Partnerschaft mbB Patentanwälte Rechtsanwälte Postfach 86 06 24 81633 München (DE)
Decision under appeal:	**Decision of the Examining Division of the European Patent Office posted on 16 December 2019 refusing European patent application No. 17173986.5 pursuant to Article 97(2) EPC.**

Composition of the Board:

Chairman	M. Müller
Members:	T. Alecu
	B. Müller

Summary of Facts and Submissions

I. The appeal is against the decision of the Examining Division. The Appellant requests that the decision of the Examining Division be set aside and that a patent be granted on the basis of the main request, or on the basis of one of two auxiliary requests.

II. The main request corresponds to the main request underlying the decision, which was refused for a lack of inventive step. The first auxiliary request corresponds to the second auxiliary request filed during the oral proceedings before the Examining Division, which was not admitted a late filed and prima facie not overcoming the inventive step objections in respect of the then first auxiliary request (see the minutes of the oral proceedings before the Examining Division, points 7 and 7.1). These two requests were re-filed with the statement of grounds of appeal (the first auxiliary request still bearing the label "Second auxiliary request claims" and the date 12 November 2019). The second auxiliary request (dated 11 January 2024) was filed in response to the preliminary opinion of the Board.

III. The decision refers inter alia to documents:

D1: Sanghamitra R et al., "*An algorithm for converting floating-point computations to fixed-point in MATLAB based FPGA design*"
D2: Anonymous, "*Floating-point arithmetic - Wikipedia*", version of 26 May 2016
D3: Marcus G et al., "*A fully synthesizable single-precision, floating-point adder/ substractor and multiplier in VHDL for general and educational use*",

D4: Anjanasasidharan et al.: "*VHDL implementation of IEEE 754 floating point unit*",
D5: Guillermo M, "*VHDL Entity FPadd.single_cycle*"

In its preliminary opinion, the Board further introduced document:

DA1: Parhami P, "*COMPUTER ARITHMETIC Algorithms and Hardware Designs*", second edition, 2010, Chapters 17 (Floating-Point Representations) and 18 (Floating-Point Operations).

IV. With its reply to the Board's preliminary opinion, the Appellant also enclosed the following documents:

Fell J: "Embedded World 2017 Awards: Innovations in hardware, software and tooling products", E+T Web article (March 15, 2017), available at https://eandt.theiet.org/2017/03/15/embedded-world-2017-awards-innovations-hardware-software-and-tooling-products
Anonymous: “Optimism Reigns At Embedded World 2017”, Silicon Semiconductor Web article (March 22, 2017), available at https://siliconsemiconductor.net/article/101681/Optimism_reigns_at_Embedded_World_2017

V. Claim 1 of the main request defines:

A method for generating code (220) for a graphical model (202), the graphical model comprising at least one block (204), wherein the at least one block comprises an operator to operate on one or more inputs, at least one of the one or more inputs having a floating point data type, the method comprising:
- interpreting at least one of the one or more first inputs as a sign, a mantissa, and an exponent, and

- generating instructions, e.g. source code and/or execution instructions, for the graphical model, the generated instructions comprising:

- instructions for the one of the one or more inputs, wherein the instructions comprise variables representing the sign, the mantissa, and/or the exponent; and

- instructions for a function that performs integer arithmetic on the variables when the instructions are executed, the function having equivalent behavior as the operator operating on the one or more inputs;

wherein the variables preserves the raw bit pattern of the one of the one or more inputs;

wherein the bit representations of the one of the one or more inputs are sliced or decomposed into the variables; and

wherein the generated code is suitable for use with programmable logic devices, application specific integrated circuits and/or microcontrollers that lack Floating Point Units.

VI. Claim 1 of the first auxiliary request defines:

A method for generating code (220) for a model (202), the model comprising at least one block (204), wherein the at least one block comprises a computational element that comprises an operator to operate on one or more inputs, at least one of the one or more inputs having a floating point data type, the method comprising:

- interpreting at least one of the one or more first inputs as a sign, a mantissa, and an exponent, and

- creating one or more in-memory intermediate representations of the model, wherein creating the one

or more in-memory intermediate representation of the model comprising;

- retrieving an intermediate representation substructure (216a) from a library (214), the intermediate representation substructure (216a) implementing functionality equivalent to the computational element of the model and using integer arithmetic;

- including the given intermediate representation substructure (216a) in the one or more in-memory intermediate representations created for the model;

- generating execution instructions for the model utilizing the one or more in-memory intermediate representations, the generated execution instructions comprising:

- instructions for the one of the one or more inputs, wherein the instructions comprise variables representing the sign, the mantissa, and/or the exponent; and

- instructions for a function that performs integer arithmetic on the variables when the instructions are executed, the function having equivalent behavior as the operator operating on the one or more inputs;

wherein the variables preserve the respective part of the raw bit pattern of the one of the one or more inputs;

wherein the bit representations of the one of the one or more inputs are sliced or decomposed into the variables; and

wherein the generated code is suitable for use with programmable logic devices, application specific integrated circuits and/or microcontrollers that lack Floating Point Units.

VII. Claim 1 of the second auxiliary request differs from that of the first auxiliary request only in the formulation of the last feature, which is in this request as follows:

wherein the generated code is Hardware Description Language (HDL) code and a programmable logic device without Floating Point Unit is synthesized based on the HDL code.

Reasons for the Decision

The application

1. The application relates to code generation from a graphic computational model, programmed in e.g. Simulink, Matlab, Simscape, or Modelica. It addresses the problem of converting such models, which use floating point numbers or expressions, to code for programmable logic devices (e.g. FPGA or ASIC) which do not include floating point units (FPUs) (description, page 6).

2. In the proposed method a floating point number is split into its three components, i.e. sign, mantissa, and exponent according to the IEEE 754 standard, represented as Boolean, integer or fixed-point data types. The graphic model is first converted into an intermediated in-memory representation by replacing the (mathematical) operations in the model with substructures (retrieved from a library) which operate on the three components of the floating point numbers using integer or fixed point arithmetic (pages 13 and 14). For example (see figure 3), a multiplication operation can be carried out by separately XORing the signs, multi-

plying the mantissas, and adding the exponents; rounding, normalization and adjust elements are necessary so that the output conforms to the IEEE standard. Other examples are provided in the application (e.g. figure 15 is a diagram for an adder/subtractor).

2.1 From this intermediate representation code is generated (see figure 6c), and deployed on a microcontroller for execution or used to synthesize a circuit on a programmable logic device (figure 7).

The prior art

3. Document D1 is concerned with providing a method for automatic code generation for devices providing only fixed-point arithmetic. It explains (in the abstract) that while designers often develop the application in Matlab, FPGA designs are "*limited to fixed-point arithmetic*". So the "*first step in a flow to map MATLAB applications into hardware is the conversion of the floating-point MATLAB algorithm into a fixed-point version using 'quantizers' from the Filter Design and Analysis (FDA) Toolbox for MATLAB*". D1 proposes an automatic conversion tool, based on the quantizer functions, which constrains the error within a specified limit.

4. D2 describes the floating-point format and explains the associated floating-point operations.

5. Document D3 teaches (see abstract) "*fully synthesizable hardware descriptions in VHDL*" for floating-point adder and multiplier operators. D3 aims to provide them for "*general and educational use*". The design is target independent (section II) and the corresponding block diagrams (figures 2 and 3) are said to be identical

with, or similar to, those in a textbook (cited as [3]).

5.1 Document D5 is the code for the adder of D3.

5.2 DA1 is an excerpt concerning floating-point arithmetic from the second edition of the textbook cited in D3.

6. D4 provides for a VHDL implementation of the IEEE 754 standard, including a set of floating-point operations, without however going much into detail.

Main request

7. The main request was refused for a lack of inventive step starting from D1 in view of D2, D3, or D4.

7.1 The Examining Division was of the opinion (see the decision, points 11.2 and 11.3) that the distinguishing features (as also enumerated by the Appellant in its statement of grounds of appeal at paragraph 4) were those related to the fact that bit representations of the input variables having floating point data type numbers are "sliced" into three different components, and that code is generated using "integer arithmetic" for use with devices lacking FPUs.

7.2 The Examining Division implicitly acknowledged (in point 12.2), as the Appellant also noted (statement of grounds of appeal paragraph 5), a further difference, namely that the initial model is a graphical one (as opposed to a "text" program, such as standard Matlab code), but was of the opinion that this did not contribute to the technical effect. The Appellant does not

challenge this last aspect and the Board agrees on it as well.

8. The Examining Division formulated the corresponding objective technical problem (in point 11.5) as "*how to generate code [...] so that [it] is suitable for use with [devices] that lack Floating Point Units, while avoiding quantization errors to keep the precision of floating point operations*". It then stated, with reference to D2, D3 and D4, that the conversion of a floating point number into the three claimed components and floating point operations using these components on standard integer ALUs were well known and therefore obvious in order to solve the stated problem.

9. In its preliminary opinion, the Board essentially agreed with the opinion of the Examining Division, stating inter alia the following (point 11):

"*the obvious solution for the generation of code from a program or model with floating point data types to a hardware platform which does not support floating point operations is to pick a representation for floating point data types and to generate code which emulates the missing floating point operations. The claimed representation is a standard one (IEEE 754), and for this representation the required code is well-known (see e.g. D2, D3 and esp. DA1, sections 17.3 and 18)*".

9.1 The Board also rejected the Appellant's allegation (statement of grounds of appeal, paragraphs 18 to 28) that the claimed integer arithmetic operations were different from those described in D2, and noted (point 13), also with reference to DA1, that the application

used the same IEEE standard representation and performed the operations in the same commonly known way.

10. In its reply to the Board's preliminary opinion (see e.g. point 15), and also during oral proceedings, the Appellant argued that the objective technical problem, though "viable", was formulated partly with hindsight, as it presupposed a FPGA without FPU.

10.1 Further, although emulation was indeed obvious (point 26 of the reply), the obvious emulation method, and the one traditionally used, was one where the floating-point representation was approximated with a fixed-point representation, which could be used on FPGAs. The invention proposed a different method, which was very positively received in the community and had even received awards, as shown by the documents enclosed with the reply to the Board's communication (see point IV above).

10.2 The approach had advantages (reply, points 31 to 35) over the allegedly traditional approach in terms of precision and ease of validation and numerical consistency.

11. The Board notes, as it already did in its preliminary opinion, that the formulated problem is not only obvious, but was also known in the art at the claimed priority date of the present application. As D1 already explains, DSP algorithm designers often work in higher-level languages, be it Matlab or anyother language providing floating-point operations. However, code may have to be generated for target devices such as FPGAs which may not include FPUs.

11.1 Emulation by fixed-point approximation will not be a satisfactory solution if it does not provide the required precision. Indeed, some applications require a representation of numbers across a wide range and high precision operations which fixed-point arithmetic does not provide but which floating-point arithmetic is known to support (see DA1 chapter 17, page 349). Emulating floating point operations based on the standard sign, mantissa, exponent representation is then the obvious solution.

11.2 The Board agrees that an actual and complete implementation of this solution may not be trivial and has no reason to doubt that the delivery of a product making it available within a complex programming environment may be prize worthy. The claims, however, are limited to the core ideas and do not contain any non-trivial implementation details.

12. Thus the Board concludes that claim 1 of the main request lacks inventive step (Article 56 EPC).

First auxiliary request

13. This request was not admitted by the Examining Division. The decision itself does not reason this point, contrary to the requirements of Rule 111(2) EPC.

14. The minutes of the oral proceedings, however, make the reasons clear (see point II above). Therefore, the Board considers the insufficiency of reasoning in the decision itself not to constitute a fundamental deficiency (Article 11 RPBA 2020) or a substantial procedural violation (Rule 103(1)(a) EPC).

15. Admittance of this request is governed by Article 12(6) RPBA 2020.

15.1 The Board remarks that the Examining Division effectively decided that this request lacks inventive step for the same reasons as the first auxiliary request underlying the decision, which had been fully discussed with the Appellant. It also accepted that all its other objections to the then first auxiliary request were overcome. Moreover, the Appellant replied to the reasons based on which the then first auxiliary request was refused.

15.2 In view of these circumstances, the Board decides to admit this request.

16. The added features relate essentially to the use of a library which contains the code for various (three component) floating-point operators and by reference to which they are integrated in the intermediate model.

16.1 According to the Appellant in its statement of grounds of appeal, the technical effect of the additional distinguishing features "*includes increased speed of generating code with increased precision compared to D1, while maintaining portability of the generated code*".

16.2 During the oral proceedings the Appellant further argued that the intermediate substructures allowed the conversion of the graphical models in a modular fashion. The generated code, i.e. the claimed intermediate representation, could not only be used on a computer, but also compiled for FPGA. The floating-point format was converted to an integer format while preserving the raw bit pattern. The Appellant referred within the description to page 14, first paragraph,

page 15, second paragraph, and to figure 3, showing an explicit intermediate step of converting the floating-point variable into an integer format. Such matter was neither disclosed not hinted at in the prior art.

17. In the Board's opinion, it is common practice in programming to use libraries of commonly used functions (as also discussed by the Examining Division at point 16). This was not contested by the Appellant. It follows from this and from the conclusion on the main request above, that providing a library to store code emulating commonly used floating point operations, and using it during code generation, is obvious to the skilled person. This accounts for the technical effects argued by the Appellant in its statement of grounds of appeal.

17.1 Moreover, the Board cannot identify any features of the claim which would support the idea that the code "is both usable on a computer and can be compiled for a FPGA", apart from the fact that it is not clear what this exactly means. If might be intended to mean that the model could be validated on the computer (because it behaves the same on the computer as it would when run on the FPGA (see also point 10.2 above), but this would be a direct consequence of using floating-point operation emulation, which the Board already found to be obvious. So this argument is not convincing.

17.2 Regarding the preservation of the raw bit pattern, the Board cannot see that difference based on the claimed features. According to the claim, the floating-point variable is converted into "*variables representing the sign, the mantissa, and/or the exponent*", on which integer arithmetic is performed. This corresponds entirely to the IEEE standard representation (see DA1

figure 17.3) and to floating point operations carried out accordingly.

18. Thus the Board concludes that claim 1 of this request lacks inventive step as well.

Second auxiliary request

19. This request was filed, according to the Appellant, as a response to the Board's concerns whether the problem was technical at all (point 10 of the preliminary opinion), which had not been discussed before. The Board accepts that the amendments are a fair attempt of addressing that point and that they do not cause any other issues. So it decides to admit this new request (Article 13 RPBA 2020).

20. However, this amendment has no impact on the discussion above. The Appellant did not argue otherwise. Claim 1 of this request lacks an inventive step as well.

Order

For these reasons it is decided that:

The appeal is dismissed.

The Registrar: The Chairman:

L. Stridde Martin Müller

Decision electronically authenticated