PROPOSAL FOR AN EFFICIENT SOFTWARE OPTIMIZATION METHOD FOR SOFTWARE-DEFINED RADIO

Yuji IKEDA*, Kosuke YAMAZAKI*, Toshiyuki MAEYAMA**, Yoshio TAKEUCHI*
*KDDI R&D Laboratories, Fujimino, Japan
**Takushoku University, Hachioji, Japan
Email: *{yj-ikeda, ko-yamazaki, takeuchi}@kddilabs.jp, **tmaeyama@es.takushoku-u.ac.jp
Topic Section Number: 15.2 Processor Concurrence, Latency, real time optimization, power optimization, memory optimization

ABSTRACT

Recently, various digital signal processors (DSP) for Software defined Radio (SDR) have been released. To develop SDR software, the processing time must be within the interval required by the wireless communication system. If this period is too short, the SDR software needs to be optimized to maximize the potential of the DSP. However, as each DSP has its own specialized hardware architecture, the software optimization takes a very long time. Moreover, the software optimized for one DSP does not work well on other DSPs.

In this paper, we propose an efficient method for optimizing the SDR software. Our proposal enables a reduction in the amount of work required to optimize the SDR software for the target DSP. In the proposed method, the information needed to execute the optimization, while taking into consideration the hardware architecture of the target DSP, is added to the target source code. The source code optimization tool (SCOT) executes optimization automatically using both the added information and the information about the characteristics and the constraints of the hardware architecture of the target DSP. Using the proposed method, all the software programmer has to do is add the information to the source code. Accordingly, the amount of work required for optimization can be reduced.

Moreover, we made a prototype of SCOT and evaluated the performance of the optimization. The results showed that by using the prototype, the processing time of several operations was reduced by about 75% from that of non-optimized source code and the work needed for optimization was reduced by about 90% compared with that of optimizing manually.

1. INTRODUCTION

The demand for high-speed and wide capacity wireless communication systems has been growing. In the 3GPP2 (3rd Generation Partnership Project 2) [1], the CDMA2000 1x EV- DO Rev. 0 was standardized in 2002 [2] and has since been very widely utilized. Meanwhile, the next generation communication system, CDMA2000 1x EV- DO Rev. A was standardized in 2006 [3]. With such rapid evolution, devices for wireless signal processing must be replaced very soon after being developed, which significantly increases the initial cost of the system. SDR can be a very effective solution to this problem. SDR is a technique which enables digital signal processing related to a wireless communication system via software alone. The SDR technique makes it possible to achieve a wireless communication upgrade via software updates only, without any hardware replacement. Because updating software is much less costly than updating hardware, both initial and maintenance costs can be reduced.

In general, the required interval for real-time operation is very short in a wireless communication system. In order to satisfy this requirement by means of software, use of a DSP is very effective. This is because in a wireless communication system, multiplication operations, such as the convolution operation, are executed numerous times. However, as each DSP has very unique characteristics, the SDR software has to be optimized in order to utilize it. For example, the number of cores, the connection method of each core and the memory structure, and so on, differ between each DSP. Moreover, some DSPs have a unique extended instruction set or an accelerator. Although software optimization is executed by the compiler, it is not enough to maximize the potential of the target DSP. Thus it is necessary to optimize the SDR software manually in consideration of the hardware architecture of the DSP.

Traditional approaches related to SDR implementation employ a variety of optimization schemes according to the target processor's architecture [4, 5, 6]. For example, in the case of the SB3010 [7], its multi-core architecture is fully taken into account. One of the effects of using multi-core architecture is parallelization of operation. In order to utilize this feature, processing is assigned to each core so that the load on each core is equalized as much as possible. In addition, memory re-assignment is also executed. SB3010 has several types of internal and external memory. Although

external memory has a large capacity, access requires many cycles. The variables and functions that are used frequently are assigned to internal memory in order to reduce the access latency.

The software optimization mentioned above takes a very long time. Moreover, the software already developed does not work well on other DSPs. Accordingly, if the target DSP is replaced, the SDR software must be optimized again. This redundant cost, related to the need to re-modify the SDR software, will occur every time a used processor is replaced. Such development costs and longer lead-time are treated as a part of the total cost for new SDR software development [8].

2. PROBLEM OF AUTOMATIC OPTIMIZATION

In order to resolve this problem, we studied an automatic software optimization method [9]. When optimizing the SDR software, it is imperative to select the required information from the target source code in order to utilize the characteristics and to consider the constraints of the DSP. This is because the information required for software optimization differs very significantly depending on the target DSP. For example, to utilize multi-core architecture, information on the minimum unit of processing which is executed by one of DSP cores is essential. Also, to use an accelerator effectively, information which processing can apply to the accelerator and what variables are inputs or outputs of the accelerator need to be determine .In order to select such information correctly from the source code, human experience and judgment are both absolutely imperative. Thus, it is very difficult to optimize SDR software automatically for the target DSP.

3. PROPOSED METHOD

To overcome this problem, we propose a method of optimizing SDR software for the DSP that is more effective compared with executing software optimization manually. Fig.1 shows the architecture of the proposed method. As this figure shows, the proposed method is characterized by the fact that the source code information is added to the non-optimized source code, which is written in a high-level language such as C or C++. The source code information is that which is needed to optimize software in order to utilize the target DSP, as mentioned in the previous section. This information is added by the software programmer because the required information differs vastly depending on the target DSP. Examples of the source code information are shown below.

- Information on a unit of processing that should be executed by an accelerator or one DSP core
- Information on the I/O variable used in the processing
- Information on the iteration number of the processing

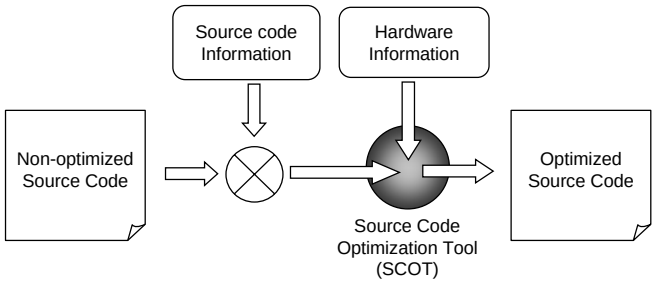

Fig 1. The architecture of the proposed method

The procedure of optimizing SDR software using the proposed method is shown below.

Step1 : Adding the source code information

First, the software programmer analyzes the non-optimized source code and selects the information taking the characteristics of the target DSP into consideration. Then, the software programmer embeds the result of analysis as the source code information within the non-optimized source code.

Step2 : Optimizing SDR software

The source code optimization tool (SCOT) executes the optimization automatically using both the non-optimized source code with the source code information and the hardware information, which together constitute the characteristics and constraints of the DSP hardware architecture. Examples of hardware information are given below.

- Information on the hardware architecture characteristics, such as whether the DSP has an accelerator or is multi-core.
- Information on whether a unique variable can be used.
- Information on the constraints of the hardware, such as the width of the bus, the number of ports, and the memory size.

In step1, it is only necessary to pick up a limited amount of information such as the processing unit or the I/O values depending on the characteristics of the target DSP. The SCOT executes optimization automatically in order to maximize its potential. By using the proposed method, all the software programmer has to do is add the source code information (step1). Thus, the amount of work can be reduced compared with manual optimization.

4. PROTOTPYE OF SOURCE CODE OPTIMIZATION TOOL

In order to demonstrate the applicability of the proposed method, we made a prototype of the SCOT.

Table 1 : Examples of the hardware information

Hardware information
S5530 has a PLD called ISEF
Customized instructions can be used in ISEF
The width of the interface of this unit is 128 bits
This unit has 3 interfaces for input and 2 for output

4.1 Hardware information

The S5530 processor manufactured by Stretch Inc [10] was used. This processor has a programmable logic device (PLD) called ISEF (Instruction Set Extension Fabric). The features of this unit are shown below.

- In this unit, a customized instruction set can be defined by users in C/C++.
- The width of the interface of this unit is 128 bits. This type of variable is called a wide register (WR). By packing certain variables into a single WR, parallelization can be realized.

In particular, parallelization of the operation using ISEF is a very effective means of reducing processing time. When passing the argument to ISEF, WR must be used.

In order to utilize the S5530, it is vital to discover the processing operations that are repeated many times by the iteration loop. Examples of hardware information are shown in Table 1. The prototype tool already contains this information.

4.2 Source code information

In the trial, tags are added to the non-optimized source code as the source code information. As described in the previous section, the key point of optimizing the S5530 is discovering the processing operations that should be executed by ISEF. Fig.2 shows an example of the non-optimized source code added the tags and Table 2 shows examples of the tags.

4.3 Behavior of the prototype tool

The prototype tool picks up the tags added to the source code and executes optimization automatically. The behavior of the prototype tool is shown below.

Step 1: Calculate how many variables can be packed into a WR

In this step, the prototype tool calculates how many variables can be packed into a WR (128bit) before optimizing the non-optimized source code. The procedures of this step are shown in Fig.3.

(1) The prototype tool searches for the tag “calc-input” and detects the processing input variable.
(2) The prototype tool searches for the tag “alloc” and extracts the data size of the input variable.

```
start

alloc   int     TempI;
alloc   int     TempQ;
alloc   short   iDataI;
alloc   short   iDataQ;
alloc   short   Tap;

for(i = 0; i < Num; i++)[

    init    TempI = 0;
    init    TempQ = 0;

    loop-init       j = 0;
    loop-condition  j < 48;
    loop-renew      j += 1;

        calc-input  iDataI = Rx_I[i + j];
        calc-input  iDataQ = Rx_Q[i + j];
        calc-input  Tap = RxTap[j];
        calc-exe    TempI += iDataI * Tap;
        calc-exe    TempQ += iDataQ * Tap;

    loop-end

    calc-output DataI[i] = TempI >> 14;
    calc-output DataQ[i] = TempQ >> 14;
]

end
```

Fig.2 : Examples of the non-optimized source code added the tag

(3) The prototype tool calculates how many variables can be packed into a WR. For example, if the type of variable which is added to the tag “calc-input” is short, eight short-type (16 bits length) variables can be packed into a single WR-type (128 bits length) variable.

Step 2: Optimizing the source code for S5530

In this step, the prototype tool picks up the tag and optimizes the non-optimized source code for S5530. Here, the optimized source codes for S5530 are composed of the codes that define customized instructions and that which call the customized instructions. Fig.4 shows the optimized source codes for S5530. The optimization procedure is shown below. Here, the number of each item corresponds to that in Fig.3.

(1) If the tag “start” is picked up, the prototype tool starts optimization.
(2) If the tag “alloc” is picked up, the variable that added this tag is defined in the source code that defines customized instructions.
(3) If the tag “init” is picked up, a customized instruction *Init-func( )*, which executes initialization, is defined and called.
(4) If the tags “loop-init,” “loop-condition,” and “loop-renew” are picked up, the description of the iteration loop is output taking into consideration that the number of repetitions which is reduced by packing a certain variable into a WR
(5) If the tag “calc-input” is picked up, a customized instruction *Calc-func()*, which executes the processing, is defined and called. Furthermore, the description of packing a certain variable into a WR is output in the

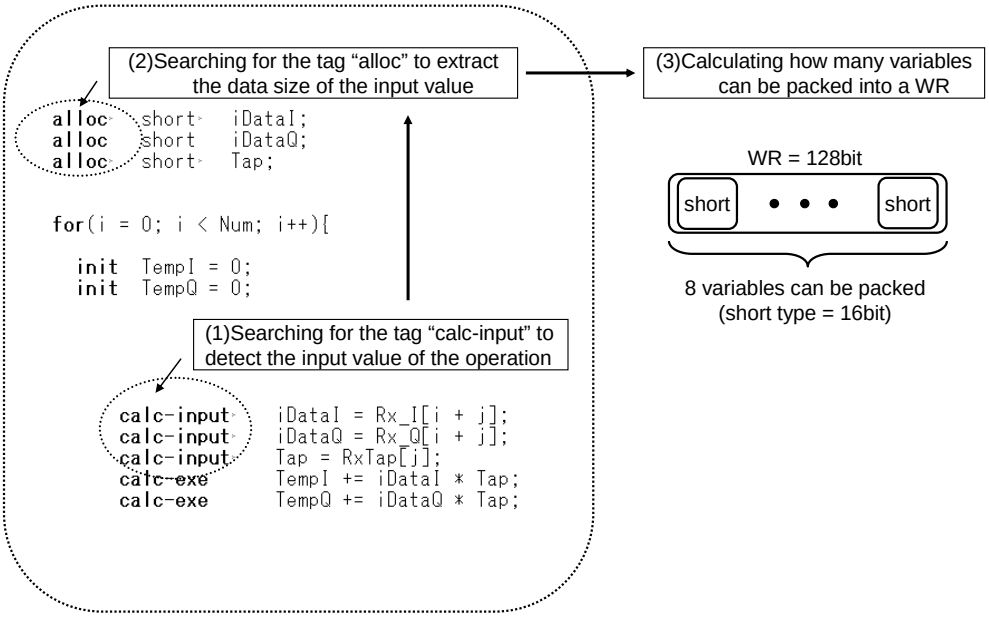

Fig.3 : The procedure of step1

source code, which calls customized instructions and the description of extracting variables from a WR in ISEF is output in the source code that defines the customized instruction. Here, if the number of “calc-input” tags exceeds 4, the prototype tool judges this processing cannot be executed by ISEF and stops optimizing because ISEF has only 3 interfaces for input.

(6) If the tag “calc-exe” is picked up, the description added this tag is copied in a customized instruction *Calc-func()*.

(7) If the tag “calc-output” is picked up, a customized instruction *Output-func()*, which executes storage of the results of the processing executed by ISEF to a WR, is defined and called. Moreover, the description of extracting this result from a WR is output in the source code, which calls the customized instructions. Here, if the number of “calc-output” tags exceeds 3, the prototype tool determines that this processing cannot be executed by ISEF and stops optimizing because ISEF has only 2 interfaces for output.

(8) If the tag “end” is picked up, the prototype tool ends the optimization.

In this trial, once the software programmer adds the tag to the non-optimized source code, the prototype tool executes optimization automatically based on the procedure described in section 4.3. Accordingly, the amount of work required for the optimization can be reduced.

5. PERFORMANCE EVALUATION

5.1 Target operations

Several operations are optimized through use of the prototype tool and can be used to evaluate performance. The target operations are shown below.

(i) Square of the absolute value

In this operation, the square of the absolute value of the complex figure is calculated. That is, if the input value is X_1 (complex figure), the output value Y_1 is calculated as below.

$$Y_1 = X_1 * X_1^*$$

Here, ” * “ means the complex conjugate.

(ii) OR circuit

In this operation, the logical addition of 3 inputs is executed. That is, if the input bits are D_1, D_2 and D_3, the output bit Y_2 is calculated as below.

$$\begin{cases} Y_2 = 0 \ : \ if \ D_1 = D_2 = D_3 = 0 \\ Y_2 = 1 \ : \ otherwise \end{cases}$$

(iii) FIR Filter

In this operation, the convolution operation of the input vector and the tap vector is executed. That is, if the input vector and the tap vectors are $\mathbf{X_2}$ and $\mathbf{L_1}$, the output vector $\mathbf{Y_3}$ is calculated as below.

$$Y_3[i] = \sum_{j=0}^{N-1} X_2[i-j] * L_1[j]$$

Here, N is the length of the tap vector.

5.2 Result of performance evaluation

Table 3 shows the results of the performance evaluation. In this table, the percentage of the reduced cycle time is shown. This figure is calculated based on how much the cycle time is reduced compared with the results without optimization and are shown as a percentage. Here, the operation "Square of the absolute value" and "OR circuit" was repeated 8192 times, and in the "FIR filter" operation, the lengths of the input vector $\mathbf{X_2}$ and the tap vector $\mathbf{L_1}$ were 8192 and 48, respectively. Moreover, the input values (X_1, $\mathbf{X_2}$, D_1, D_2, D_3, $\mathbf{L_1}$) are the short type, so the eight parallel operations can be realized in ISEF.

As this table shows, the performance of the prototype tool is a little worse than that of the optimization by the software programmer regardless of the type of operation. This is mainly because the prototype tool developed in this paper is an abridged edition, and it picks up the tags and only replaces the description for S5530. In the case of "optimization by the software programmer", the software programmer codes while taking the location of each description into consideration so that the code can be executed more effectively. If the prototype tool is developed further, the difference in the cycle time, "optimization by the prototype tool" and "optimization by the software programmer" may be reduced.

On the other hand, by using the proposed method, the amount of work required to optimize the source code is reduced by about 90% compared with "optimization by the software programmer". In optimization by the software programmer, the amount of time required was about one or two hours. In the optimization using the prototype tool, the amount of time required for adding the tag was about 10 minutes. In terms of the work required for the optimization, it can be said that the proposed method is very efficient.

Table 3: Results of the performance evaluation

	Percentage of the reduced cycle time [%]	
	Optimization by the prototype tool	Optimization by the software programmer
Square of absolute value	72	77
OR circuit	79	83
FIR filter	71	80

6. CONCLUSION

This paper presents a method of optimizing the software for SDR. In the proposed method, the source code information that is needed to execute optimization, taking the hardware architecture of the target DSP into consideration, is added to the target source code. The SCOT executes the optimization automatically in order to utilize the potential of the DSP using both the source code information and the hardware information which means the characteristics and constraints of the hardware architecture of the DSP. Using the proposed method, the amount of work required for the optimization can be reduced because all the software programmer has to do to optimize the SDR software is to add the source code information. In order to demonstrate the applicability of the proposed method, we made a prototype tool and evaluated the performance of the optimization. In the trial, the processor used was the S5530, and the tags were added to the target source code as the source code information. The results showed that by using the prototype tool, processing time for several operations was reduced drastically compared with that of non-optimization. Moreover, compared to manual optimization, the reduction in cycle time ranged from about 5%-10%. On the other hand, the amount of work required to optimize the source code is reduced by about 90% compared with the optimization by the software programmer. Considering the drastic reduction of work, the proposed method is very efficient. In particular, it is very effective for DSPs that have an accelerator or a PLD. This is because we only have to select the part that should be executed by the accelerator or PLD and then add sufficient source code information.

On the other hand, for DSPs that have multiple-core architecture, we should also consider the composition of the software in order to assign the processing to each core effectively. And for some processing, synchronization of each core may be required. Such optimization is more complex, so a method for optimizing DSPs having multiple-core architecture will be proposed in the future.

7. REFERENCES

[1] 3rd Generation Partnership Project 2 : 3GPP2, "http://www.3gpp2.org"

[2] 3GPP C.S0024-0 Ver.4.0 "cdma2000 High Rate Packet Data Air Interface", Oct.2002

[3] 3GPP C.S0024-A Ver.3.0 "cdma2000 High Rate Packet Data Air Interface", Sept.2006

[4] Gweon-Do JO, Min-Joung SHEEN, Seung-Hwan LEE, and Kyoung-Rok CHO, ``A DSP-Based Reconfigurable SDR Platform for 3G Systems, " IEICE Trans. Communications, Vol. E88-B No. 2, pp. 678-686, February 2005.

[5] Bob KRAFT, "A high-performance SDR Implementation for Compact-PCI", Compact PCI Systems Product Guide, July–August 2002.

[6] Andrew DULLER, Daniel TOWNER, Gajinder PANESAR, Alan GRAY and Will ROBBINS, ``picoArray technology: the tool's story", Design, Automation and Test in Europe, 2005, pp.106–111 Vol.3

[7] Daisuke KAMISAKA, Singo WATANABE, and Yoshio TAKEUCHI, ``Study on Software Optimization for Software Defined Radio using Multiple-core DSP", GSPx 2006.

[8] Tokihiko YOKOI, Yoshimitsu IKI, Jun HORIKISHI, Katsuji MIWA, Yoshio KARASAWA, Akira FUKUDA, Jun-ichi TAKADA, Yuichi KURODA, Takayasu SHIOKAWA, Yukitsuna FURUYA, Shigenari SUZUKI, Yasuhiro SENBA,

Yoshihide YAMADA, Hiroshi HARADA, Yasuo SUZUKI and Hiyomichi ARAKI “Software Receiver Technology and Its Applications (Special Issue on Software Defined Radio and Its Technologies)”, IEICE transactions on communications 2000 Vol.83 Num.6 p.1200–1209, 2000 June

[9] Kosuke YAMAZAKI, Shingo WATANABE and Yoshio TAKEUCHI, ``A Study of Software Portability for Software Defined Radio, '' IEICE Society Conference, B17-19, 2007.

[10] Stretch Inc, http://www.stretchinc.com

Table 2 : Examples of the tags

tags	purpose
start,end	Indicating the part that should be executed by ISEF
alloc	Indicating the variable that is used in a customized instruction
init	Indicating the variable that should be initialized
loop-init	Indicating the initial value of the iteration loop
loop-condition	Indicating the condition whether the iteration loop is continued
loop-renew	Indicating the condition of the renewing of the value
loop-end	Indicating the end of the iteration loop
calc-input,calc-output	Indicating the I/O variables of ISEF
calc-exe	Indicating the position of the calclation

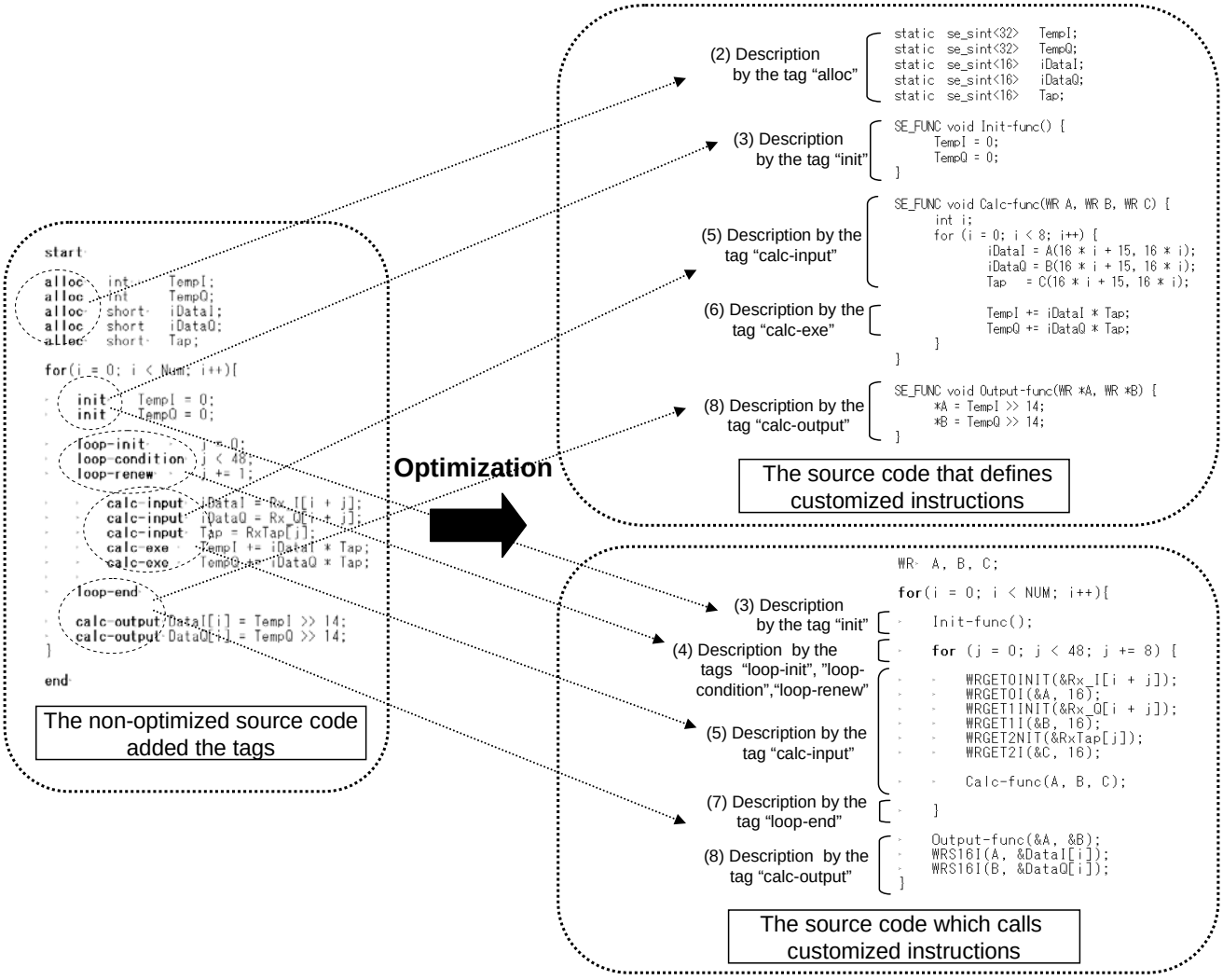

Fig.4 : The optimized source code for S5530