Tadokoro et al. *EPJ Quantum Technology* (2025) 12:81
https://doi.org/10.1140/epjqt/s40507-025-00384-9

EPJ Quantum Technology
a SpringerOpen Journal

RESEARCH Open Access

Quantum algorithm compiler for architectures with semiconductor spin qubits

Masahiro Tadokoro[1,2*], Ryutaro Matsuoka[1] and Tetsuo Kodera[1]

*Correspondence:
tadokoro.m.5768@m.isct.ac.jp
[1]Institute of Science Tokyo, 2-12-1 Ookayama, Meguro-ku, Tokyo, 152-8552, Japan
[2]Mizuho–DL Financial Technology Co., Ltd., 2-4-1 Kojimachi, Chiyoda-ku, Tokyo, 102-0083, Japan

Abstract

Various architectures have been proposed using a large array of semiconductor spin qubits with high-fidelity and high-speed gate operation. However, no quantum algorithm compilers have been developed which can compile quantum algorithms in a consistent manner for the various architectures, limiting the discussion on evaluating the efficiency of quantum algorithm implementation. Here, we propose Qubit Operation Orchestrator considering qubit Connectivity and Addressability Implementation (QOOCAI), a first quantum algorithm compiler designed for various architectures with semiconductor spin qubits. QOOCAI can compile quantum algorithms to various architectures with different qubit connectivity and addressability, which are important features that affect the efficiency of quantum algorithm implementation. Furthermore, we compile multiple quantum algorithms on different architectures with QOOCAI, showing that higher qubit connectivity and addressability make the algorithm implementation quantitatively more efficient. These findings are crucial for developing semiconductor spin qubit devices, highlighting QOOCAI's potential for improving quantum algorithm implementation efficiency across diverse architectures.

Keywords: Quantum computing; Quantum circuit; Semiconductor spin qubits; Quantum algorithm compilation; Qubit connectivity; Qubit addressability; Two-dimensional qubit array

1 Introduction

Semiconductor spin qubits are promising candidate systems for high-fidelity and high-speed gate operation [1–10], and the scalability of qubits utilizing existing mature microfabrication techniques [11–14]. So far, various architectures with large arrays of semiconductor spin qubits have been proposed [15–18], and when considering the implementation of quantum algorithms on them, qubit connectivity and addressability [18] are important features. High qubit connectivity allows for the elimination of the pre/post process to compensate for the connection of qubits in multi-qubit gate implementations. It contributes to the implementation efficiency of quantum algorithms. On the other hand, qubit addressability is the accessibility to each qubit. It is assumed that good qubit addressability allows to operate multiple quantum gates in the same time step, which also makes the

implementation of quantum algorithms more efficient. However, there has been no compiler which can compile quantum algorithms in a consistent manner for architectures with different qubit connectivity and addressability. Therefore, the requirements for these architectural features in implementing quantum algorithms have not been discussed so far.

Here, we propose a quantum algorithm compiler, Qubit Operation Orchestrator considering qubit Connectivity and Addressability Implementation, named QOOCAI. This compiler consists of three processes required when implementing quantum algorithms for architectures and can efficiently implement various quantum algorithms according to the qubit connectivity and addressability of the architecture. In addition, QOOCAI can be used to compile algorithms for different architectures in a consistent manner. In this paper, we present quantitative evaluation results of the efficiency of algorithm implementation when multiple algorithms are implemented for three architectures with different qubit connectivity and addressability. These results represent an evaluation of semiconductor spin qubit architectures in terms of quantum algorithm implementation and suggest a fundamental and critical method for future architectural considerations.

2 Background

Quantum algorithm compilers to implement quantum algorithms for architectures have been discussed in each qubit system in recent years. This section provides an overview of these researches and summarizes the position of this research.

Quantum algorithm compilers generally make compiled quantum circuits by sequentially applying three processes: gate decomposition, qubit routing/mapping, and operation scheduling [19]. Since the output circuit clearly shows which quantum gates should be implemented in which order, it is possible to calculate the input logical quantum circuit by performing qubit operations based on this output circuit. The following is an overview of these three processes.

2.1 Gate decomposition

In the first process, gate decomposition, quantum gates on logical quantum circuits are decomposed into quantum gates that can be implemented by the architecture. For example, in semiconductor spin qubit systems, it is possible to implement any 1qubit gate, but the implementation of multi-qubit gates may require decomposition. In particular, the fidelity of multi-qubit gates of 3 or more qubits is low in semiconductor spin qubit systems so far [20], so here we decompose them into 2 and 1qubit gates. This process should be considered in any qubit system, and methods for decomposing arbitrary quantum gates into 1qubit gates and CNOT gates have been proposed in previous research [21]. It is possible to apply such methods for semiconductor spin qubit systems.

2.2 Qubit routing/mapping

Next, in the second process, qubit routing/mapping, an appropriate qubit connection completion process like SWAP gates [22] or qubit shuttling [23–26] is inserted to implement multi-qubit gates on logical quantum circuits (qubit routing). In addition, since it is more efficient to implement a multi-qubit gate locating at the beginning of the quantum circuit without any extra qubit routing process, it is better to consider which logical qubit corresponds to which physical qubit at the same time as the qubit routing process (qubit mapping). This process needs to be considered for qubit systems with qubit connectivity

limitation. Although some qubit routing/mapping methods have been proposed for specific semiconductor spin qubit architectures systems [19, 27, 28], there is no method that can be applied to multiple architectures. On the other hand, methods have been developed for superconductor qubit systems with two-dimensional qubit arrays like semiconductor qubit systems [29–31]. However, these methods assume that there is a one-to-one correspondence between logical and physical qubits, and that physical qubits have direct connection with each other. Therefore, these methods may not be directly applicable in the case assuming architectures in which physical qubits exist sparsely on the quantum dot (QD) array and have no direct connection, as proposed for semiconductor spin qubit systems [17]. QOOCAI avoids this problem by applying the concept of virtual physical qubits between the logical qubit and the physical qubit (discussed details in Sect. 3).

2.3 Operation scheduling

Finally, the third is operation scheduling. Here, the order of quantum gate implementation is determined by the qubit addressability of the architecture. In this process, if multiple quantum gates can be implemented at the same time, they are implemented simultaneously for efficiency. On the other hand, some architectures have limitations on simultaneous implementation due to the resonance frequency design or wiring design. Therefore, it is necessary to determine operations with consideration of these unique features of the architecture. As the qubit addressability is different in each qubit system, it is necessary to develop methods for efficient operation scheduling in each system. In semiconductor spin qubit systems, ESR [1, 2] and EDSR [3–5] are used to implement 1qubit gates by applying RF magnetic fields or RF electric fields to the architecture with a resonance frequency of the qubits. However, especially in view of the effects of crosstalk between qubits, the resonance frequency must be chosen with a degree of separation in the limited frequency bandwidth available for operation, and in future large-scale qubit arrays it is expected that multiple qubits will share the same frequency. In this situation, implementing the same 1qubit gate for multiple qubits sharing the same frequency has the advantage that it can be implemented simultaneously. On the other hand, if 1qubit gates are required to be implemented for only some of the qubits sharing the same frequency, each qubit cannot be operated independently, resulting in a qubit addressability limitation. Although an operation scheduling method has been proposed for a specific semiconductor spin qubit architecture with such a qubit addressability limitation [19], there is no consistent method for operation scheduling for different architectures. Therefore, we propose a novel operation scheduling method for different architectures with qubit addressability limitation (discussed details in Sect. 3).

In summary, regarding the process of quantum algorithm compilers for semiconductor spin qubit architectures, it is possible to use prior methods for gate decomposition. However, it is necessary to correspond the logical qubits and architectures although prior methods have been proposed for qubit routing/mapping. Also, novel methods for operation scheduling need to be considered. Here we propose QOOCAI, a quantum algorithm compiler for semiconductor spin qubit architectures that overcomes these challenges.

3 Methods

This section presents the details of the processing content in QOOCAI and the architecture used for the evaluation.

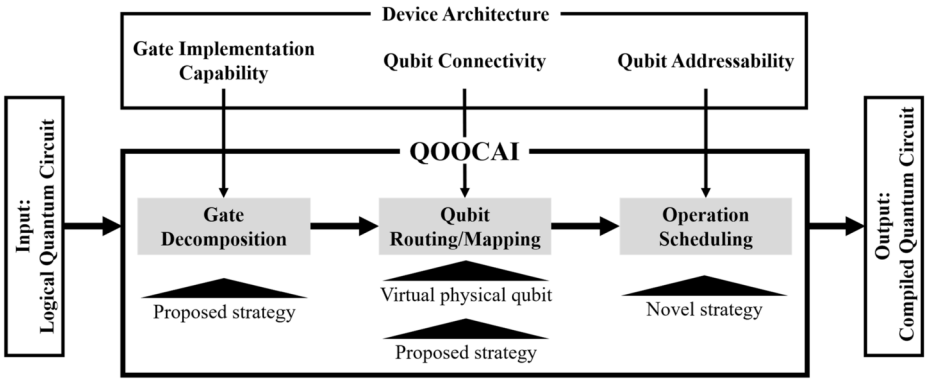

Figure 1 Overall scheme of QOOCAI

3.1 QOOCAI

Figure 1 shows the overall scheme of QOOCAI proposed in this paper. QOOCAI includes the processes of gate decomposition, qubit routing/mapping and operation scheduling. In gate decomposition, the methods of previous studies are used as plug-ins. Here, we assume the strict architectural requirement that all quantum gates are decomposed into arbitrary 1qubit gates and CZ gates. As mentioned in Sect. 2, QOOCAI makes two main contributions: first, it enables the application of qubit routing/mapping methods to architectures without a one-to-one correspondence between logical and physical qubits, and second, it introduces a new operation scheduling method. Each of these is discussed in more detail below.

Firstly, for the qubit routing/mapping method, the concept of virtual physical qubits is introduced to enable proposed methods to be applied. As proposed in [17], an architecture has been proposed in which sparsely spaced spins are arranged in a large QD array with spin shuttling. In such architectures, not all QDs store spins, but only some of them. For the spins to interact, they need to be shuttled to the QD pairs with which they can interact. This architecture has advantages such as reducing the effects of crosstalk and making the wiring structures required for control more efficient. However, in considering the implementation of the algorithm, the previously proposed qubit routing/mapping methods assume a one-to-one correspondence between logical and physical qubits and cannot be directly applied to these architectures. QOOCAI therefore avoids this problem by introducing the concept of virtual physical qubits. A virtual physical qubit is a set of QDs that is virtually treated as one physical qubit and includes one spin (the physical qubit) and surrounding multiple QDs. Since there is a one-to-one correspondence between logical qubits and virtual physical qubits, and the virtual physical qubits are connecting to each other, it is possible to apply qubit routing/mapping methods proposed in previous research. Although various approaches can be taken to determine the set of QDs of virtual physical qubits depending on the arrangement of QDs and spins in the architecture, we assume that the architecture has a symmetric arrangement of QDs and spins as proposed in [17]. Symmetrical architectures are expected to have high uniformity as the spins are equally spaced and the effect of crosstalk is homogenized. Also, from a control point of view, it has the advantage that the architecture can be easily scaled. As such a symmetrical architecture is formed by repeatedly arranging unit structures, these unit structures

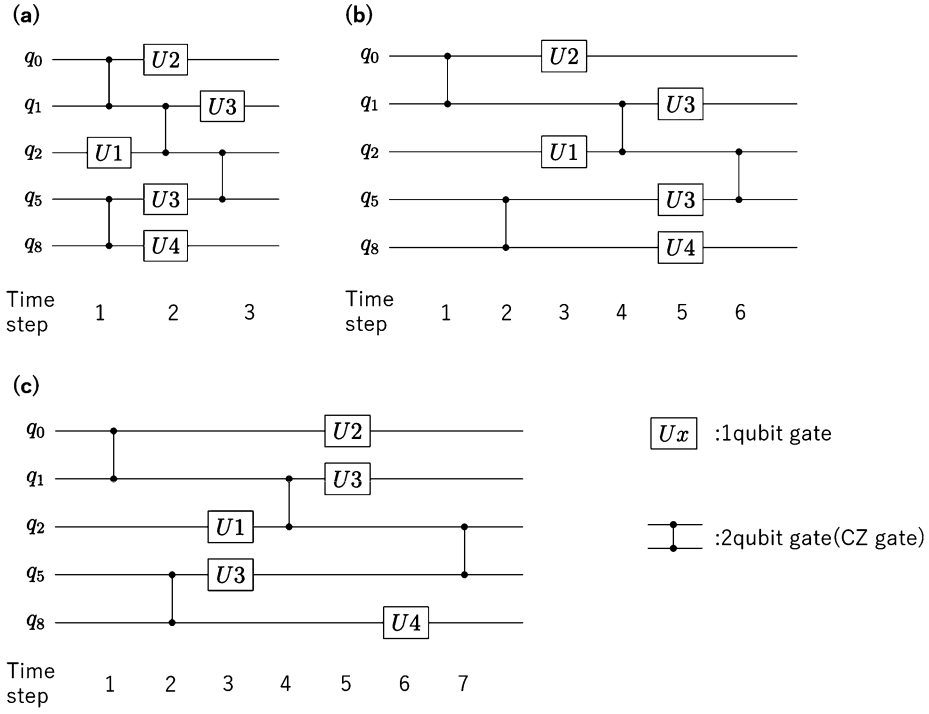

Figure 2 Example of scheduling of a logical quantum circuit in the architecture which allows only two types of 1qubit gates to be operated simultaneously. (a) Example of a logical quantum circuit to be implemented. U1, U2, U3, and U4 are 1qubit gates with different parameters, and quantum gates connected by vertical lines are 2qubit gates (CZ gates). (b) An example of efficiently scheduling. Since only two types of 1qubit gates can be operated at the same time step, Time step 3 and 5 are used for 1qubit gate implementations. (c) An example of inefficient scheduling of (a). If U1 and U3 are scheduled in Time step 3, it is necessary to schedule three types of 1 qubit gates (U2, U3, and U4) in a later time step, requiring an additional time step for 1qubit gates

can be regarded as virtual physical qubits. Examples of specific virtual physical qubits are given in *Architectures to be evaluated*.

Next, for the operation scheduling process, we developed a novel scheduling method that takes qubit addressability into account. There are constraints on the simultaneous operation of multiple types of 1qubit gates on architectures with qubit addressability limitations, because multiple QDs have the same resonance frequency. For example, let us consider implementing the logical quantum circuit shown in Fig. 2(a) to the architecture which allows only two types of 1qubit gate to be operated simultaneously. This circuit consists of five 1qubit gates of four different types. The most efficient scheduling is shown in Fig. 2(b), in which two types of 1qubit gates are operated at the same time. In contrast, if the combination of 1qubit gates to be operated simultaneously is not appropriate, the time step required for implementation increases as shown in Fig. 2(c).

Here, depending on the architecture, it may be possible to schedule multiple 2qubit gates to the same time step, such as the two 2qubit gates scheduled to Time step 1 and 2 in Fig. 2(b). This possibility varies depending on the metal gate structure that controls the exchange interaction between spins [2], and here we assume that multiple 2qubit gates cannot be equally scheduled to the same time step. However, this method is valid even when multiple 2qubit gates can be scheduled at the same time step.

Similarly, it may be possible to schedule both 1qubit gates and 2qubit gates at the same time step. When implementing 1qubit gates, the spins need to be well confined in the QD, whereas when implementing 2qubit gates, the QD confinement needs to be weakened in order to strengthen the exchange interaction between the spins [2, 7–9]. The capability of simultaneous implementation of these two operations depends on the design of the metal gate structure of the architecture [15–18]. Here, we separate the time step to schedule 1qubit gates and 2qubit gates.

Algorithm 1 shows a method for efficiently performing such scheduling. In this method, scheduling is performed by repeating the following steps.

1. Getting front layer

   First, the quantum gates to be scheduled are acquired. Here, quantum gates in front three layers among the unscheduled quantum gates are acquired. The layer is the set of quantum gates whose operations do not conflict on logical quantum circuits. For example, in Fig. 2(a), each time step corresponds to a layer. Normally, only the scheduling of the most front layer should be considered, but in this case, multiple layers need to be acquired because simultaneous operations across layers are considered as shown in Fig. 2(b). The acquired layers are called 1st layer, 2nd layer, and 3rd layer, in order from the most front layer. This process is necessary in every loop, as the content of the quantum gates in the front layer changes each time the scheduling proceeds.
2. Scheduling 2qubit gates in the 1st layer

   Next, the 2qubit gates in the acquired 1st layer are scheduled. In considering the scheduling of multiple 1qubit gates in 1st and 2nd layer to the same time step, like in Time step 3 and 5 in Fig. 2(b), the scheduling of 2qubit gates in 1st layer that should be scheduled before those 1qubit gates is operated here. Using Fig. 2(a) as an example, the CZ gate with q0 and q1, and the CZ gate with q5 and q8 correspond. Here, if 2qubit gates can be scheduled in the same time step, they are treated as such; if there is a constraint, they are scheduled in separated time steps. In addition, this process does not change the order of 2qubit gate implementation of the quantum circuit, so there is no need to implement qubit routing/mapping processes again after this process.
3. Scheduling 1qubit gates

   3.1 Scheduling 1qubit gates in the 1st layer

   Following the 2qubit gate, the 1qubit gates in the 1st layer are first scheduled. U1 in q2 corresponds to the example in Fig. 2(a).

   3.2 Scheduling additional same 1qubit gates in the 2nd layer

   Since the scheduling of 1qubit gates in the 1st layer has been completed up to the previous step, the scheduling of the 2nd layer can be considered in the subsequent scheduling.

   Among 1qubit gates in the 2nd layer, 1qubit gates with the same type as the 1qubit gate that was scheduled in the previous step are scheduled. These 1qubit gates can operate simultaneously with the 1qubit gate scheduled in the previous step, since the scheduling of the 2qubit gate have been completed. In Fig. 2(a), for example, there is no U1 in the 2nd layer, so there is no corresponding quantum gate.

   3.3 Scheduling additional other 1qubit gates in 2nd layer

Moreover, if the resonance frequency to be used for operation is left over in the scheduling of 1qubit gates up to this point, it is possible to schedule another 1qubit gate simultaneously. In such a case, the 1qubit gate with a different type from the 1qubit gate that has been scheduled up to now can be scheduled in advance from the 2nd layer. In this case, however, it is preferable that the 1qubit gates to be scheduled are different types from the 1qubit gate in the 3rd layer. This is to avoid deepening time steps by scheduling the 1qubit gates in the 2nd and 3rd layers, which could be scheduled at the same time step, by separating them.

Note that in this process, it is not possible to schedule the 1qubit gate in the 3rd layer to the time step in which the 1qubit gate in the 1st layer is scheduled. The qubit in which the 1qubit gate of the 3rd layer is operated must have a 2qubit gate operated in the 2nd layer, and the 1qubit gate of the 3rd layer cannot be scheduled until after the scheduling of this 2qubit gate. Since the 2qubit gate in the 2nd layer cannot be scheduled at the process of scheduling the 1qubit gate in the 1st layer, it is only necessary to consider scheduling the 1qubit gate in the 2nd layer at this point.

Taking Fig. 2(a) as an example, the resonance frequency used for operation is 2 (meaning that two types of 1qubit gates can be operated simultaneously), and since U1 has already been scheduled, one more type of 1qubit gate can be scheduled. Therefore, one of U2 in q0, U3 in q5, or U4 in q8 can be scheduled, but since U3 is included in the subsequent layer (Time step 3) in this case, U2 in q0 or U4 in q8 are scheduled in priority over U3 in q5. In Fig. 2(b), U2 in q0 is scheduled.

4. Fixing the Schedule

   The contents of the scheduling done up to this point are fixed. In particular, the scheduling of 1qubit gates should be done so that the number of 1qubit gate types that can be operated simultaneously in each time step is not exceeded. The process described above is repeated until all unscheduled quantum gates are eliminated. This allows efficient scheduling to be planned while satisfying the simultaneous operation constraints of the architecture.

3.2 Architectures to be evaluated

QOOCAI can compile quantum algorithms appropriately for each architecture with different qubit connectivity and addressability. Therefore, it is possible to evaluate the efficiency of implementing quantum algorithms for each architecture under the same conditions. Here, we evaluated the efficiency of implementing quantum algorithms for three architectures with different qubit connectivity and addressability by estimating the computational cost of the compiled quantum algorithm using QOOCAI. The three architectures assumed in this paper are named Square_2ch (Fig. 3(a)) [17, 19], Square_3ch (Fig. 3(b)), Triangle_3ch (Fig. 3(c)) [32, 33]. The architectures follow the naming convention "(QD array lattice) _ (number of resonance frequency)" and have different qubit connectivity and addressability from each other. In these architectures, QDs, indicated by circles, are arranged on a square or triangle lattice, in which spins, indicated by arrows corresponding to physical qubits, are periodically confined. Although spins can move through the QDs, the coupling between QDs is limited to only the most neighboring QDs, which means that

Algorithm 1 QOOCAI's operation scheduling procedure

Input: list of gates to schedule *input_gates_list*, the number of resonance frequencies in architecture *n_f*

Output: list of gates scheduled in order of execution *scheduled_gates_list*

while *input_gates_list* is not empty**:**

```
    gates_list = []
    # 1 Getting front layers
    1st layer, 2nd layer, 3rd layer = Get gates in first/second/third front layer of input_gates_list
    # 2 Scheduling 2qubit gates in the 1st layer
    gates_list.append(2qubit gates in 1st layer)
    # 3 Scheduling 1qubit gates
    # 3.1 Scheduling 1qubit gates in the 1st layer
    gates_list.append(1qubit gates in 1st layer)
    # 3.2 Scheduling additional same 1qubit gates in the 2nd layer
    1qubitgates_2nd layer =1qubit gates in 2nd layer
    for gate in 1qubitgates_2nd layer:
        if gate in 1qubit gates in 1st layer:
            1qubitgates_2nd layer.remove(gate)
            gates_list.append(gate)
    # 3.3 Scheduling additional other 1qubit gates in the 2nd layer
    n_uniq = Get the number of unique 1qubit gates in 1st layer
    n_remain = n_uniq % n_f
    for n_remain:
        if unique_gates in 1qubitgates_2nd layer unincluded in 3rd layer:
            gates_list.append(unique_gates)
        else:
            unique_gates = Get unique 1qubit gates in 1qubitgates_2nd layer
            gates_list.append(unique_gates)
    # 4 Fixing the schedule
    scheduled_gates_list.append(gates_list)
    input_gates_list.remove(gates_list)
```

these architectures have a qubit connectivity limitation. Moreover, the two or three resonance frequencies (red, blue, and green) are shared by QDs, and the architecture has a qubit addressability limitation that prevents individual control of each qubit.

There are various metrics that can be used to estimate the computational cost of a compiled quantum algorithm, we use the simple method of adding up the number of time steps required for each gate operation. The sum of the operation times of each time step is the time required to implement the entire circuit, and the product of the operation fidelity of each time step is the circuit fidelity, using the Estimated Success Probability concept of [19]. In other words, the evaluation metric of the number of time steps corresponds to the factor of the execution time and fidelity of the quantum circuit. Here, we focus only on features related to qubit connectivity and addressability of the architecture, and the impact of surrounding structures to realize these features such as metal gate arrangements,

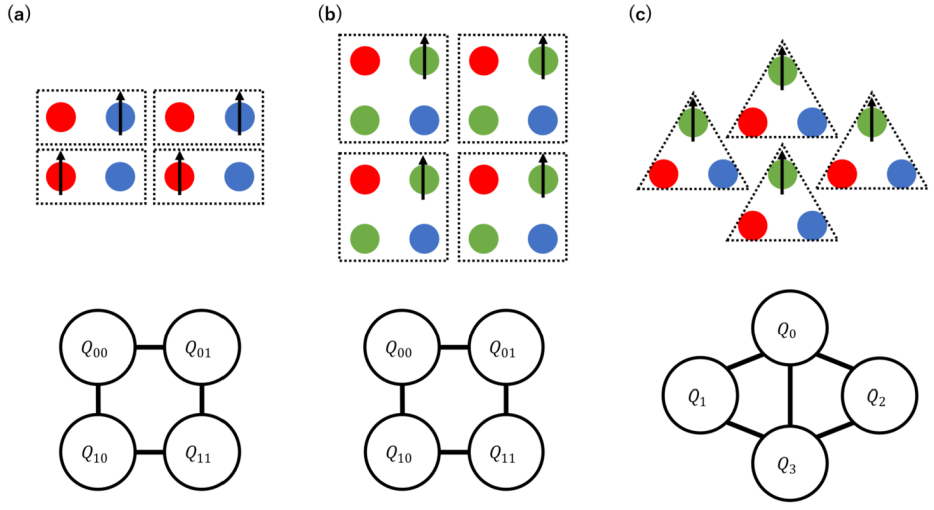

Figure 3 The architecture of (a) Square_2ch, (b) Square_3ch, and (c) Triangle_3ch, where the efficiency of quantum algorithm implementation is evaluated. The circles on the upper side of each figure indicate the QDs, and their colors correspond to resonance frequencies. Spins (corresponding to physical qubits) indicated by the arrows are periodically arranged. Bottom side figures represent virtual physical qubits ($\boldsymbol{Q}$), which correspond to QDs and spins surrounded by the dotted lines on upper figures

stripline designs for ESR, and micromagnet designs for EDSR [34] is not included in the evaluation. The estimation of circuit fidelity or execution time using operation fidelity or operation time, for example, is greatly affected by these peripheral structures. Also, it is difficult to make an objective evaluation of these evaluation metrics since there are currently no demonstration experiments of the operation fidelity or time of a large-scale semiconductor spin qubit architecture. By naively adding up the number of time steps as metrics, we can evaluate only the efficiency of the implementation of quantum algorithms brought about by the qubit connectivity and addressability of the architecture. In addition, for each time step, further reduction may be expected depending on the architecture due to these surrounding structures [17], but this is not the subject of discussion here.

4 Results

Here we present the results of implementation efficiency evaluation of multiple quantum algorithms for three architectures: Square_2ch, Square_3ch, and Triangle_3ch.

First, conditions used in this numerical experiment is described. For gate decomposition and qubit routing/mapping in QOOCAI, we used Unroller, SabreSwap, and SabreLayout [29] in the Qiskit (Ver 0.44) library. In gate decomposition, it was assumed that only arbitrary 1qubit gates and CZ gates could be implemented, given the stricter requirements of the architecture. For operation scheduling, we used scheduling method that takes qubit addressability into account, as described above. The quantum algorithm used in the evaluation is obtained from RevLib [35]. All experiments in this paper are executed on a server with Apple M2 Max CPU (12 logical cores) and 32 GB memory. The Operating System is macOS Sonoma 14.4. All experimental code is run in python 3.10.

4.1 Evaluation of QOOCAI

Here are the results of evaluation. First, to confirm that QOOCAI compiles the quantum algorithm efficiently, we compared the number of time steps when the quantum algorithm

is compiled by QOOCAI and the baseline compiler to be compared. It would have been desirable to make a comparison with the method of operation scheduling, which is widely applicable to semiconductor spin qubit architectures. However, since there is no quantum algorithm compiler other than QOOCAI that can compile quantum algorithms for multiple architectures, the baseline compiler which uses a simpler method compared to QOOCAI is designed and compared in this paper.

The following is a description of baseline compiler processing. The baseline compiler consists of three processes like QOOCAI: gate decomposition, qubit routing/mapping, and operation scheduling. First, gate decomposition uses the Unroller included in the Qiskit library, as well as QOOCAI, to decompose all quantum gates into only 1qubit gates and CZ gates. Next, in the qubit routing/mapping process, the qubit mapping process maps logical qubits from end to end according to the order of virtual physical qubits. In the qubit routing process, all spins are basically kept in their initial placement. When 2qubit gate is operated, SWAP gates are inserted so that target two spins are moved to the adjacent QD each time and return to the initial placement after the 2qubit gate operation is completed. Finally, in the operation scheduling process, all quantum gate operations are implemented one time step at a time. Therefore, the number of time steps after compilation is equal to the number of quantum gates, which means that this process does not improve the compilation efficiency at all.

Table 1 shows the result of the number of time steps when compiling the quantum algorithms using QOOCAI for the Square_2ch architecture, and the ratio to the result compiled with the baseline compiler. Here, n is the number of qubits in each quantum algorithm, t_u, t_{CZ}, and t_{SWAP} are time steps required to implement the 1qubit gate, CZ gate, and SWAP gate, respectively, and t_{tot} is the time step for the entire quantum algorithm. In this experiment, t_{CZ} and t_{SWAP} are equal to the number of CZ gates and SWAP gates in the quantum circuit respectively, because all 2qubit gates are scheduled with separate time steps as described above. The percentages in each result indicate the ratio to the result in baseline compiler. 100% means that the compiling result by QOOCAI and the baseline compiler is the same.

The results show that QOOCAI can reduce the timestep for t_u and t_{SWAP} compared to the baseline compiler, which results in a reduction of the overall timestep, and thus QOOCAI has succeeded in improving the efficiency of quantum algorithm compilation. The reduction in t_u is due to a reduction in the number of time steps required for implementation by a novel method in operation scheduling, while t_{SWAP} is due to a reduction in the number of SWAP gates by applying the qubit routing/mapping method.

As for CZ gate, as mentioned above, both QOOCAI and the baseline compiler use 2qubit gates with separate time steps for scheduling, so t_{CZ} has the same value for all algorithms.

Looking at the differences in efficiency among the algorithms, the efficiency of 1qubit gate and SWAP gate implementation in the ising model simulation is particularly remarkable. In the Ising model simulation, since 2qubit gates between specific 2 qubits are repeatedly operated, the contribution of SWAP gate implementation efficiency due to the qubit routing/mapping process is larger than that of other quantum algorithms [29]. In addition, when the number of 2qubit gates in a quantum circuit is small, the number of 1qubit gates in a layer is assumed to be large, and it is possible in operation scheduling that the scheduling of 1qubit gates has also become more efficient as the SWAP gate implementation becomes more efficient. Figure 4 shows a scatter plot of the sum of t_{CZ} and t_{SWAP}

Table 1 Compilation results with QOOCAI and its efficiency compared to baseline compiler (Square_2ch)

	n	t_u	t_{CZ}	t_{SWAP}	t_{tot}
decod24-v2_43	4	26 (*87%*)	19 (*100%*)	4 (*29%*)	49 (*78%*)
4mod5-v1_22	5	13 (*87%*)	10 (*100%*)	1 (*25%*)	24 (*83%*)
mod5mils_65	5	17 (*85%*)	14 (*100%*)	3 (*23%*)	34 (*72%*)
4gt13_92	5	19 (*90%*)	16 (*100%*)	3 (*21%*)	38 (*75%*)
alu-v0_27	5	17 (*71%*)	15 (*100%*)	3 (*23%*)	35 (*67%*)
sym6_145	7	1290 (*84%*)	1182 (*100%*)	300 (*10%*)	2772 (*49%*)
rd73_252	10	2466 (*80%*)	2310 (*100%*)	568 (*12%*)	5344 (*54%*)
sqn_258	10	4697 (*77%*)	4618 (*100%*)	1103 (*15%*)	10418 (*58%*)
sym9_193	10	41292 (*79%*)	39296 (*100%*)	8061 (*12%*)	88649 (*56%*)
9symml_195	10	41292 (*79%*)	39296 (*100%*)	8061 (*12%*)	88649 (*56%*)
z4_268	11	1194 (*80%*)	1140 (*100%*)	305 (*17%*)	2639 (*60%*)
cycle10_2_110	12	10005 (*82%*)	9094 (*100%*)	1627 (*11%*)	20726 (*58%*)
rd84_253	12	10153 (*80%*)	9537 (*100%*)	2169 (9%)	21859 (*48%*)
radd_250	13	1270 (*81%*)	1199 (*100%*)	313 (9%)	2782 (*45%*)
adr4_197	13	1310 (*80%*)	1248 (*100%*)	363 (*17%*)	2921 (*58%*)
rd84_142	15	100 (*55%*)	133 (*100%*)	28 (*14%*)	261 (*51%*)
misex1_241	15	1926 (*80%*)	1830 (*100%*)	500 (*10%*)	4256 (*45%*)
ising_model_10	10	1081 (*36%*)	2000 (*100%*)	0 (0%)	3081 (*47%*)
ising_model_13	13	1128 (*29%*)	2600 (*100%*)	109 (5%)	3837 (*44%*)
ising_model_16	16	1300 (*27%*)	3200 (*100%*)	0 (0%)	4500 (*42%*)
Ising_model_20	20	1378 (*23%*)	4000 (*100%*)	0 (0%)	5378 (*39%*)
qft_10	10	139 (*84%*)	105 (*100%*)	30 (*16%*)	274 (*59%*)
qft_13	13	233 (*86%*)	174 (*100%*)	53 (*15%*)	460 (*58%*)
qft_16	16	340 (*83%*)	264 (*100%*)	91 (*15%*)	695 (*55%*)
qft_20	20	532 (*84%*)	410 (*100%*)	132 (*14%*)	1074 (*54%*)

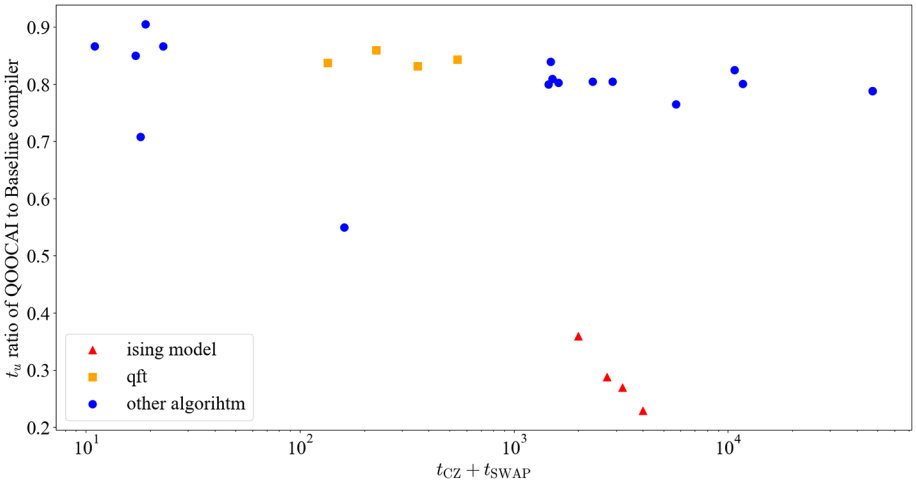

Figure 4 The scatter plot with results of each algorithm, with the sum of $\boldsymbol{t_{\mathrm{CZ}}}$ and $\boldsymbol{t_{\mathrm{SWAP}}}$ on the horizontal axis and the ratio of $\boldsymbol{t_u}$ in QOOCAI and the baseline compiler compilations on the vertical axis. The horizontal axis is the logarithmic axis, and the red triangles, orange squares, and blue circles show the results of ising model simulation, qft, and the other algorithms respectively

and the ratio of t_u by QOOCAI compared to the baseline compiler for each algorithm. Overall, the implementation efficiency by QOOCAI tends to improve as the time step of the 2qubit gate including the SWAP gate increases, and therefore better implementation efficiency is expected when compiling larger quantum algorithms.

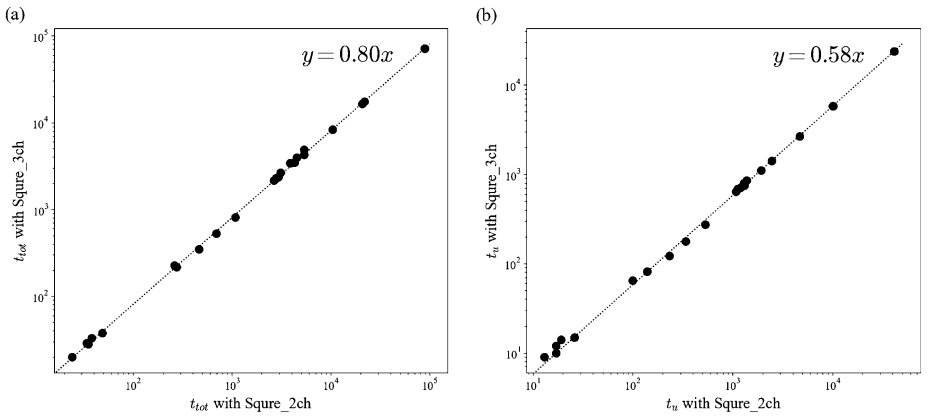

Figure 5 The scatter plot of (a) $\boldsymbol{t_{tot}}$ and (b) $\boldsymbol{t_u}$ with Square_2ch and Square_3ch. Each axis is the logarithmic axis. The dotted line is a straight line regressed linearly by the least squares method

4.2 Cross-architecture comparison

Since QOOCAI can compile quantum algorithms on architectures with different qubit connectivity and addressability using the same method, it is possible to discuss the efficiency of quantum algorithm implementation across architectures. The findings obtained from this comparison suggest the ideal architecture for implementing quantum algorithms and will be an important guideline for the future development of architectures. Here we compare the number of time steps when compiling the quantum algorithm using QOOCAI for the three architectures Square_2ch, Square_3ch, and Triangle_3ch, and evaluate the difference in efficiency of quantum algorithm implementation due to qubit connectivity and addressability of each architecture.

First, we show the difference between the results of compiling for Square_2ch and Square_3ch: Fig. 5(a) shows the plots of t_{tot} when the same algorithm is compiled into Square_2ch and Square_3ch by QOOCAI. These plots appear to be linearly correlated, and indeed, each plot is generally located on a linear regression line using the least squares method. The slope at this time is about 0.8, which indicates that the use of Square_3ch reduces t_{tot} by about 20% compared to the use of Square_2ch.

The architectural difference between Square_2ch and Square_3ch is the difference in qubit addressability. Square_2ch can implement only one type of 1qubit gates at the same time step, while Square_3ch can implement two types of 1qubit gates at the same time step. Therefore, this difference in t_{tot} is brought by t_u. Figure 5(b) plots t_u when the same algorithm is compiled into Square_2ch and Square_3ch by QOOCAI. The slope of the linear regression line using the least squares method is about 0.58, which indicates that the use of Square_3ch in this experiment can reduce t_u by roughly 42% compared to the use of Square_2ch. This result suggests that a wide range of quantum algorithms can benefit from the same t_u reduction effect by using QOOCAI, regardless of the number of qubits or quantum gates.

Next, the differences in results between the implementations for Square_3ch and Triangle_3ch are shown. Figure 6(a) shows a plot of t_{tot} and a linear regression line using the least squares method when the same algorithm was compiled for Square_3ch and Triangle_3ch by QOOCAI. The slope of the regression line is about 0.95, which indicates that t_{tot} can be reduced by 5% by using Triangle_3ch compared to Square_3ch.

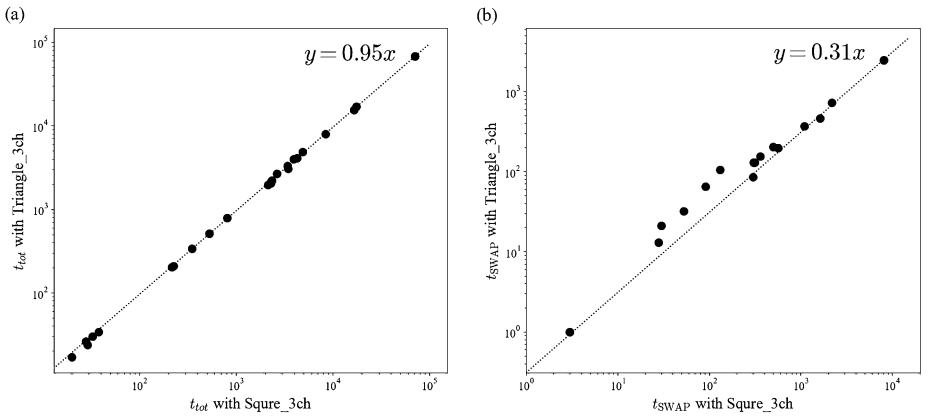

Figure 6 The scatter plot of (a) $\boldsymbol{t_{tot}}$ and (b) $\boldsymbol{t_{SWAP}}$ with Square_3ch and Triangle_3ch. Each axis is the logarithmic axis. The dotted line is a straight line regressed linearly by the least squares method

The architectural difference between Square_3ch and Triangle_3ch lies in the difference in qubit connectivity, where the number of SWAP gates can be reduced because the number of connections between virtual physical qubits increases in Triangle_3ch. Therefore, this t_{tot} difference is caused by t_{SWAP}. Figure 6(b) shows a plot of t_{SWAP} and its linear regression line when the same algorithm is compiled into Square_3ch and Triangle_3ch by QOOCAI. The slope of the linear regression line is about 0.31, which indicates that the use of Triangle_3ch reduces t_{SWAP} by roughly 79% compared to the use of Square_3ch in this experiment. This result suggests that a wide range of quantum algorithms can enjoy the same t_{SWAP} reduction effect by using QOOCAI, regardless of the number of qubits or quantum gates. Here, the reason why the reduction of t_{tot} is limited despite the large reduction of t_{SWAP} is that the number of SWAP gates in the quantum circuit used in this experiment is small compared to the entire number of quantum gates. However, assuming a quantum circuit that requires even more qubits, it may require a larger number of SWAP gate implementations. In this condition, the reduction of t_{SWAP} may have a significant impact on the efficiency of the overall quantum circuit implementation.

5 Discussion and conclusions

In this paper, we proposed a quantum algorithm compiler, QOOCAI, which can implement quantum algorithms for various architectures. By introducing the concept of virtual physical qubits in the qubit routing/mapping process, QOOCAI can plug the proposed qubit routing/mapping method in even in architectures with sparse spins on the QD array, as proposed for semiconductor qubit systems. In addition, the originally developed operation scheduling method enables efficient scheduling even in architectures where multiple QDs have the same resonance frequency and there are restrictions on the simultaneous operation of 1qubit gates. We also evaluated the implementation efficiency of multiple quantum algorithms compiled on three architectures with different qubit connectivity and addressability using QOOCAI. We obtained quantitative results showing that the implementation efficiency of the quantum algorithms is higher for architectures with higher qubit connectivity and addressability. The linearity of the results suggests that QOOCAI may have a stable time step reduction effect even for quantum algorithms with more qubits or quantum gates.

On the other hand, there are still work to be done on the compilation of quantum algorithms for practical semiconductor spin qubit systems. Here we present two of them.

The first is to further improve compilation efficiency. In QOOCAI, gate decomposition, qubit routing/mapping, and operation scheduling are executed independently and in sequence, aiming for a compiler that can be applied to a wider range of architectures. However, in the case of architectures that allow simultaneous operation of 2qubit gates or simultaneous operation of 2qubit gates and 1qubit gates, the qubit routing/mapping process should be optimized so that they can be scheduled at the same time in the operation scheduling process. Also, if each quantum gate has a different gate fidelity and gate time due to the non-uniformity of the qubits, there is room for more sophisticated compilation processes that give priority to implement quantum gates with higher gate fidelity and shorter gate time. Since discussion of these factors requires not only the arrangement of QDs and resonance frequencies, but also surrounding structures such as metal gate arrangements, development of a compilation method that assumes a specific architecture is required [19]. Another issue is the large scale and high computational cost of the optimization problem, which requires simultaneous optimization of qubit routing/mapping and operation scheduling processes.

The second is to further examine the efficiency of implementation. In this paper, we experiment the implementation efficiency of 25 quantum algorithms on three architectures. However, considering that practical quantum algorithms require the implementation of larger quantum circuits, it is necessary to examine the implementation efficiency of a wider range of quantum algorithms on a wider range of architectures to obtain general insight regarding the qubit connectivity and addressability required for the architectures. Particularly, in semiconductor spin qubit systems, there will be a trade-off between the improvement of qubit connectivity and addressability and the complexity of device fabrication to ensure qubit operation in each QD. How much qubit connectivity and addressability are required in this trade-off is an issue to be considered in the future.

These two issues are important for the future development of semiconductor spin qubit devices from the viewpoint of the efficient quantum algorithm implementation. The proposed QOOCAI and the evaluation of the efficiency of quantum algorithm implementation across architectures are the fundamental method for considering these issues and are important proposals for the development and utilization of semiconductor spin qubit devices in the future.

Abbreviations

QOOCAI, Qubit Operation Orchestrator considering qubit Connectivity and Addressability Implementation; QD, Quantum Dot.

Acknowledgements

This work was supported financially by JST Moonshot R&D Grant Number JPMJMS2065, MEXT Quantum Leap Flagship Program (MEXT QLEAP) Grant Number JPMXS0118069228, and JSPS KAKENHI Grant Numbers JP23H05455, and JP23K17327.

Author contributions

M.T. contributed to develop the compiler and evaluate it. M.T. wrote the main manuscript text with input from all authors. R.M. critically reviewed the manuscript and assisted with revisions. T.K. supervised the project.

Funding information

This work was supported financially by JST Moonshot R&D Grant Number JPMJMS2065, MEXT Quantum Leap Flagship Program (MEXT QLEAP) Grant Number JPMXS0118069228, and JSPS KAKENHI Grant Numbers JP23H05455, and JP23K17327.

Data availability
The datasets were generated from RevLib (https://www.revlib.org/). The datasets analyzed during the current study are available from the corresponding author on reasonable request.

Declarations

Competing interests
The authors declare no competing interests.

Received: 27 March 2025 Accepted: 17 June 2025 Published online: 01 July 2025

References

1. Veldhorst M, et al. An addressable quantum dot qubit with fault-tolerant control-fidelity. Nat Nanotechnol. 2014;9:981.
2. Veldhorst M, et al. A two-qubit logic gate in silicon. Nature. 2015;526:410.
3. Takeda K, et al. A fault-tolerant addressable spin qubit in a natural silicon quantum dot. Sci Adv. 2016;2:e1600694.
4. Yoneda J, et al. A quantum-dot spin qubit with coherence limited by charge noise and fidelity higher than 99.9%. Nat Nanotechnol. 2018;13:102.
5. Takeda K, et al. Optimized electrical control of a Si/SiGe spin qubit in the presence of an induced frequency shift. npj Quantum Inf. 2018;4:54.
6. Yoneda J, et al. Coherent spin qubit transport in silicon. Nat Commun. 2021;12:4114.
7. Noiri A, et al. Fast universal quantum gate above the fault-tolerance threshold in silicon. Nature. 2022;601:338.
8. Xue X, et al. Quantum logic with spin qubits crossing the surface code threshold. Nature. 2022;601:343.
9. Mills AR, et al. Two-qubit silicon quantum processor with operation fidelity exceeding 99%. Sci Adv. 2022;8:eabn5130.
10. Noiri A, et al. A shuttling-based two-qubit logic gate for linking distant silicon quantum processors. Nat Commun. 2022;13:5740.
11. Zwerver AMJ, et al. Qubits made by advanced semiconductor manufacturing. Nat Electron. 2022;5:184.
12. Philips SGJ, et al. Universal control of a six-qubit quantum processor in silicon. Nature. 2022;609:919.
13. Borsoi F, et al. Shared control of a 16 semiconductor quantum dot crossbar array. Nat Nanotechnol. 2023;19:21.
14. Neyens S, et al. Probing single electrons across 300-mm spin qubit wafers. Nature. 2024;629:80.
15. Vandersypen LMK, et al. Interfacing spin qubits in quantum dots and donors—hot, dense, and coherent. npj Quantum Inf. 2017;3:34.
16. Veldhorst M, et al. Silicon CMOS architecture for a spin-based quantum computer. Nat Commun. 2017;8:1776.
17. Li R, et al. A crossbar network for silicon quantum dot qubits. Sci Adv. 2018;4:eaar3960.
18. Tadokoro M, et al. Designs for a two-dimensional Si quantum dot array with spin qubit addressability. Sci Rep. 2021;11:1.
19. Paraskevopoulos N, et al. SpinQ: compilation strategies for scalable spin-qubit architectures. ACM Trans Quantum Comput. 2023;5:1.
20. Takeda K, et al. Quantum error correction with silicon spin qubits. Nature. 2022;608:682.
21. Iten R, et al. Quantum circuits for isometries. Phys Rev A. 2016;93:032318.
22. Sigillito AJ, et al. Coherent transfer of quantum information in a silicon double quantum dot using resonant SWAP gates. npj Quantum Inf. 2019;5:110.
23. Yoneda J, et al. Coherent spin qubit transport in silicon. Nat Commun. 2021;12:4114.
24. Noiri A, et al. A shuttling-based two-qubit logic gate for linking distant silicon quantum processors. Nat Commun. 2022;13:5740.
25. Bertrand B, et al. Fast spin information transfer between distant quantum dots using individual electrons. Nat Nanotechnol. 2016;11:672.
26. Jadot B, et al. Distant spin entanglement via fast and coherent electron shuttling. Nat Nanotechnol. 2021;16:570.
27. Escofet P, et al. Compilation Techniques for Spin Qubits in a Shuttling Bus Architecture. arXiv:2502.06263.
28. Crawford O, et al. Compilation and scaling strategies for a silicon quantum processor with sparse two-dimensional connectivity. npj Quantum Inf. 2023;9:13.
29. Li G, et al. Tackling the qubit mapping problem for NISQ-era quantum devices. In: Proceedings of the twenty-fourth international conference on architectural support for programming languages and operating systems. 2019.
30. Deng H, et al. Codar: a contextual duration-aware qubit mapping for various NISQ devices. In: 2020 57th ACM/IEEE design automation conference. 2020.
31. Zhang C, et al. Time-optimal qubit mapping. In: Proceedings of the 26th ACM international conference on architectural support for programming languages and operating systems. 2021.
32. Tadokoro M, et al. Pauli spin blockade in a silicon triangular triple quantum dot. Jpn J Appl Phys. 2020;59:SGGI01.
33. Acuna E, et al. Coherent control of a triangular exchange-only spin qubit. arXiv:2406.03705.
34. Yoneda J, et al. Robust micromagnet design for fast electrical manipulations of single spins in quantum dots. Appl Phys Express. 2015;8:084401.
35. Wille R, et al. RevLib: an online resource for reversible functions and reversible circuits. In: 38th international symposium on multiple valued logic (ismvl 2008). 2008.

Publisher's note