DUAL-PORT SRAMs YIELD BIT-SLICE DESIGNS WITHOUT MICROCODE

APPLICATION NOTE AN-09

By David C. Wyland

Abstract

High-performance controller designs use bit-slice components for their speed and design flexibility. Speeds of 10-20 million instructions per second (MIPS) are common and the designer can use bit-slice design flexibility to perform speed-critical operations in one instruction. Bit-slice designs have the drawback, however, of requiring microcode design for their implementation, often with a long development cycle. The problem is that the microcode resides in a separate, stand-alone control memory which prevents use of the kind of interactive prototyping and debugging tools associated with conventional microprocessors. The problem can be eliminated by using a dual-port SRAM for the control memory, making it part of the data memory address space, and converting the controller to a CPU by borrowing some techniques from Reduced Instruction Set Computer (RISC) designs. The result is a RISC controller where the microinstructions of the bit-slice approach become the instructions of a computer. The design approach provides all the speed and architectural flexibility of microcoded bit-slice designs, while allowing the use of interactive debugging methods associated with microprocessors.

Bit-Slice Versus RISC Architectrues

An example of a typical bit-slice controller design is shown in Figure 1. It consists of a control flow section and a data flow section. The control flow section has a microinstruction counter and the control memory. The data flow section has a register and ALU element—the bit-slice—plus a data memory and I/O registers on a data bus. Note that the control and data memories are separate. The use of separate data and instruction memories is called the Harvard architecture. The separate control memory provides some of the speed associated with bit-slice designs because it operates in parallel with the data memory. This allows the next microinstruction to be fetched from the control memory while data for the current instruction may be read from the data memory. This contrasts with conventional microprocessors which alternately get instructions and data from the same memory. This use of a single memory for instructions and data is called the Von Neumann architecture.

There is a remarkable similarity between the bit-slice controller block diagram and a block diagram of a typical RISC CPU, comparing Figures 1 and 2. The difference is that the control memory and the data memory

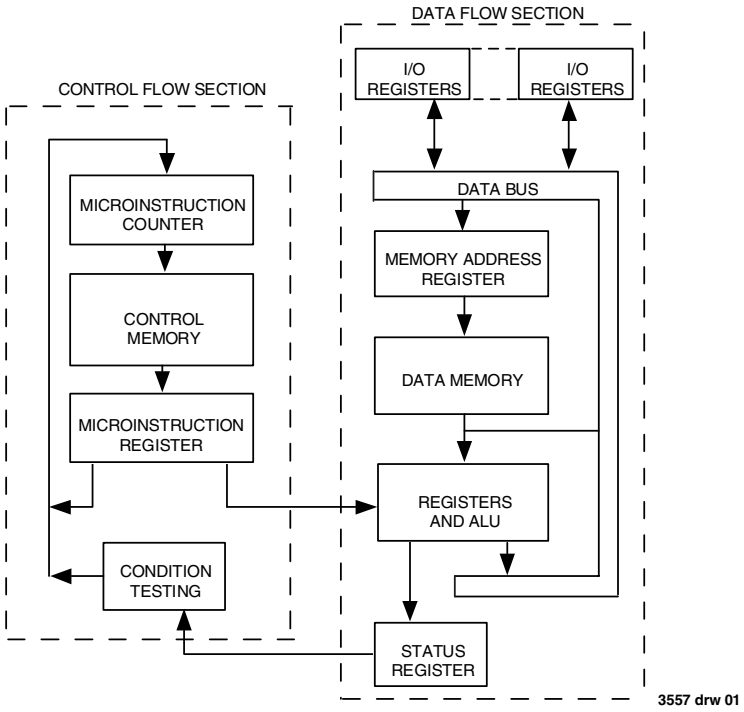


Figure 1. Bit-Slice Controller Block Diagram

MARCH 1999

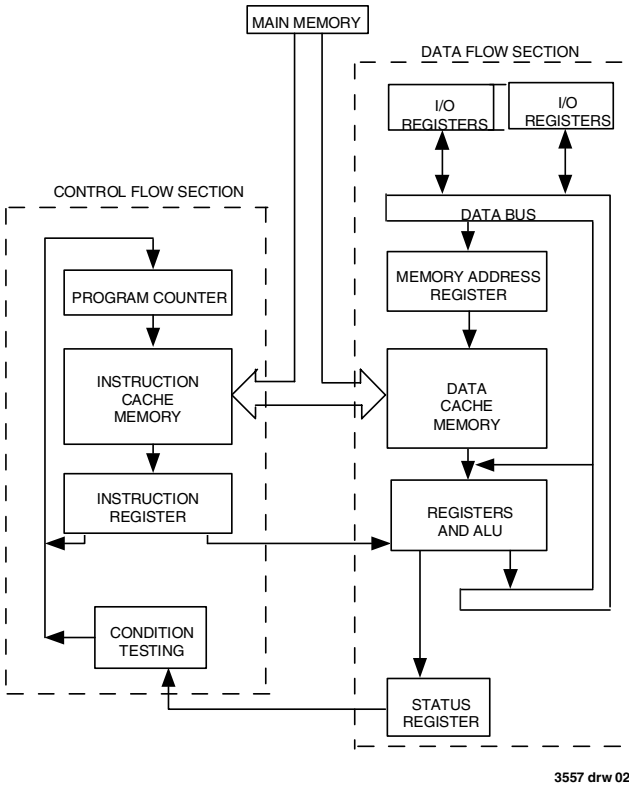


Figure 2. RISC CPU Block Diagram

of the controller have been replaced by an instruction cache memory and a data cache memory in the RISC CPU. The instruction and data cache memories work the same as their microcode counterparts except that they both contain copies of data in the common main memory. The programmer sees a single memory—the main memory—while the hardware works as if it has two independent memories. In this manner, the RISC computer has the speed advantage of the Harvard architecture and the single memory for programs and data of the Von Neumann architecture.

The instruction and data caches of the RISC architecture are equivalent to having two ports on one memory. We can apply this concept to bit-slice controllers by using a high-speed dual-port memory in place of the cache memories, as shown in Figure 3. The dual-port SRAM allows the instruction and data ports to be active simultaneously and independently, while providing both sides access to a common set of SRAM cells. Since both ports are working from the same memory, the data flow section can load and move both data and instructions in the same manner as a conventional microprocessor. As a result, this design functions as a conventional CPU with a long instruction word. This allows conventional interactive software tools, such as interpreters and monitors, to be used in system development and debugging

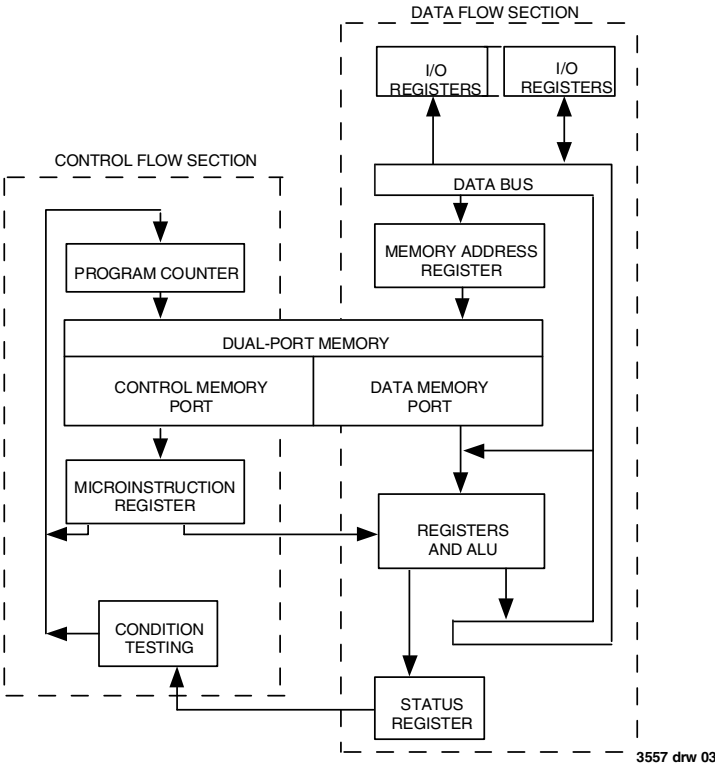


Figure 3. Bit-Slice Controller With Dual-Port Control Store

Design of a RISC Controller

The design of a RISC controller using a dual-port control memory is similar to a conventional bit-slice design except for inclusion of a minimum set of operations for a CPU. This allows use as a conventional computer for software coding and debugging. In ordinary bit-slice controller designs, the minimal CPU operation set already exists as a subset of the data flow and control operations already present.

A minimal set of CPU operations, suitable for bit-slice designs, can be derived from the instruction set of a RISC-like computer such as the Data General Nova minicomputer. It is a useful example because it is a 16-bit general register design having approximately 20 instructions and three addressing modes, yet is fully functional as a computer. From its instruction set, the list of 21 operations shown in Table 1 can be derived as a representative minimum working set. If the design includes these operations, it will function as a CPU.

Table 1. Minimal CPU Instruction Set

1.	Load register from memory at immediate address (address in instruction).
2.	Load register from memory at address in A register.
3.	Store register to memory at immediate address (address in instruction).
4.	Store register to memory at address in a register.
5-11.	Move/combine registers: move, negate, invert, add, subtract, AND, OR.
12-13.	Shift: rotate left through sign, rotate right through sign.
14.	Read status register.
15.	Write status register.
16.	Jump absolute: load program counter with immediate address.
17.	Jump register: load program counter with register contents.
18-20.	Jump absolute conditional: if zero result, if sign, if carry.
21.	Jump and save return (Program Counter) in a register.

This instruction set assumes a set of general purpose registers (typically 16 or more in bit-slice designs), a memory which contains both instructions and data and a status register which records the result of register-to-register operations. I/O registers are assumed to be mapped into the memory space so that separate instructions for them are not required.

Some of the above operations are automatically included in bit-slice controllers as a result of straightforward design. The register combination operations are provided by the bit-slice RALUs and the jump operations are commonly required as part of the control flow design. All that is required to complete the set is the ability to transfer registers to and from memory, to save and restore the status register and to save the Program Counter in a register in Jump and Save Return instructions.

Figure 4 shows a block diagram of a general purpose bit-slice controller design, based on the RISC controller architecture in Figure 3, and capable of implementing the minimal instruction set. This is a 16-bit controller design using an IDT49C402 16-bit RALU and a 64-bit instruction word. The control flow section is fully pipelined for maximum speed and uses a simple counter as the Program Counter (PC). As a result, branch execution is delayed by one instruction: the instruction following the branch is executed before the branch takes effect. This method allows maximum speed in the control flow section and is commonly used in RISC designs. A path is provided from the PC to the data inputs of the IDT49C402 for saving the PC in a register during Jump and Save Return operations. Also shown in the block diagram is an initial-load EPROM. This EPROM holds the non-volatile copy of the program to be loaded at power up. A power up flip-flop and some sequencing logic cause the contents of this EPROM to be loaded into the SRAM at power up.

In the design in Figure 4, the instructions and data share the same

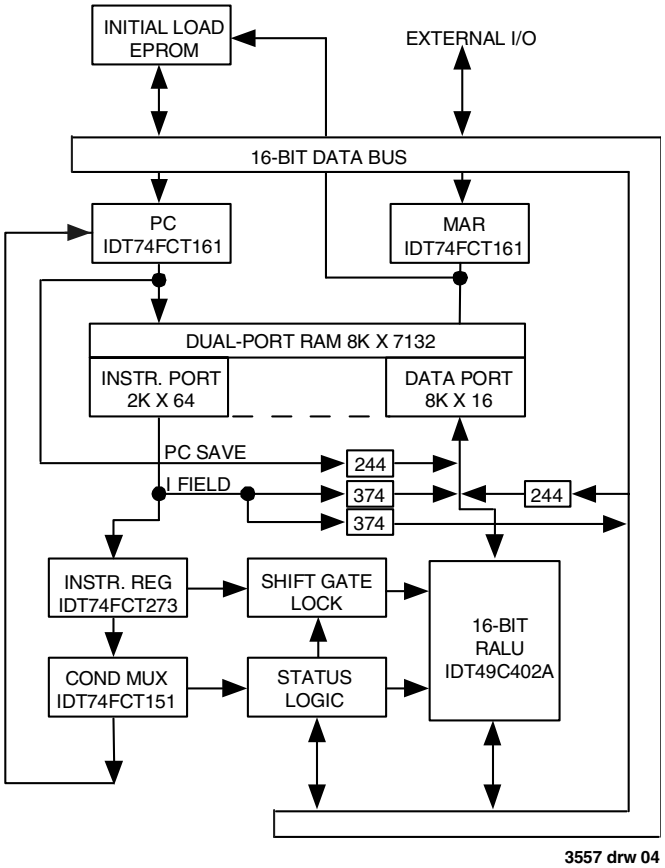


Figure 4. Dual-Port Bit-Slice RISC Controller Design Block Diagram

memory. The mapping for instructions and the mapping for data are different, however, as is shown in Figure 5. The eight dual-port RAMs are mapped as 2K words of 64 bits/word on the instruction port and as 8K words of 16 bits/word on the data port. Each 64-bit instruction word corresponds to four sequential 16-bit data words. The instruction at address 0000 on the instruction port corresponds to locations 0000, 0001, 0002 and 0003 on the data port. On the instruction port, all eight chips are enabled, resulting in 64 bits of instruction output. Only the upper 14 bits of the PC are used to address the RAM so that the address in the PC is consistent with the addressing on the data side. On the data port, the least significant two bits of the address in MAR select the appropriate 16-bit word by selecting the chip enable for the appropriate one of four pairs of dual-port SRAMs.

RISC Controller Instruction Format

The 64-bit instruction word is shown in Figure 6. Fifty of the 64 bits are used to control the basic data and control flow of the controller and 14 bits are available as additional control bits for the specific controller application. Each 64-bit instruction word from the control port of the RAM is mapped as four 16-bit words on the data memory port. A larger instruction word can be used in the same manner as in microcoded designs. It is convenient if the word width is a power of two, such as 64 or 128 bits, so that there are no gaps in the memory space as seen from the data flow side.

The IDT49C402 is controlled by the A and B fields, I_{0-9}, C_N, Stat Enable field and the Shift Gating field. The A and B fields provide the 6-

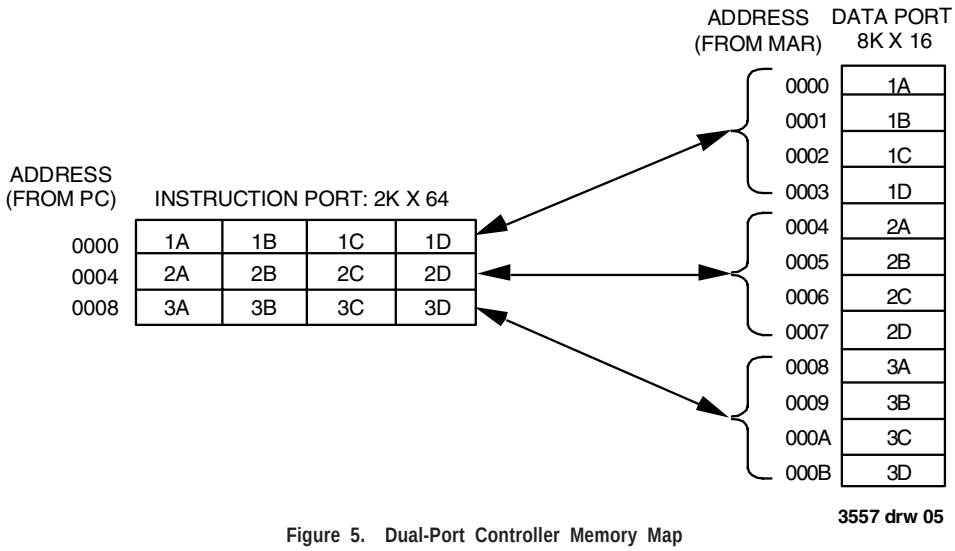
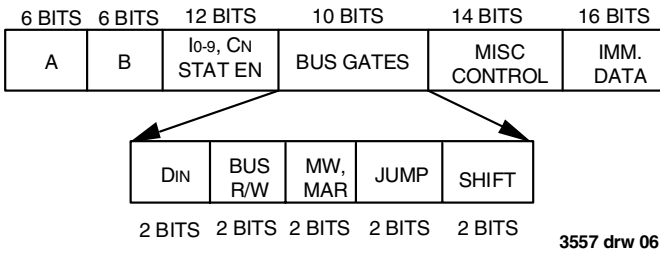


Figure 5. Dual-Port Controller Memory Map



FIELD	FUNCTION
A	402 reg address, bus read select, or jump condition select
B	402 reg address or bus write select
I_{0-9}	49C402 instructions + carry-in
Stat E_N	Enable Status reg load
D_{IN}	402 D Bus: Memory, PC. Bus, 1 field
Bus R/W	Gate Bus read @ A, write @ B
MW, MAR	Memory write enable, ld MAR enable
Jump	Enable PC load, enable condition test
Shift	402 shift/rotate gating
Imm Data	Immediate Data - address, etc.
Misc Control	Misc bits for controller functions

3557 tbl 01

Figure 6. Dual-Port Controller Instruction Format

bit addresses for the A and B register inputs on the IDT49C402. The I0-9, CN and Stat EN field provide the 10 control bits to the IDT49C402, the carry-in bit and a status register load enable, respectively, and the Shift Gating field controls the shift-in/shift-out gating for shift operations. The data source for the DIN pins of the IDT49C402 is selected by the DIN field. This field can choose the data bus, the immediate data field or the PC as the data source.

The data bus is controlled by the A and B fields as well, which provide 6-bit select codes for bus read and write operations, respectively, and by the bus read/write, memory write and load MAR bits. The default operation is to gate the data from the IDT49C402 onto the data bus. The load MAR and memory write bits allow writing this data into the memory and/or MAR from the bus. The bus read bit disables the IDT49C402 outputs and gates an I/O register onto the bus as determined by the 6-bit A field. The bus write bit causes bus data to be written into an I/O register selected by the B field.

Branch operations are controlled by the Jump and A fields. The Jump field enables loading of the PC from the bus, which is the branch operation. The A field provides the 6-bit condition select code for conditional branch operations.

The Misc Control field provides 14 bits for direct control of additional devices. This field would typically be used for gates and strobes to additional devices such as parallel multipliers, FIFOs, disk controller chips and other devices which communicate with, and are controlled by, the RISC controller.

Implementing Minimal Instruction Set

The RISC controller design must now be checked to ensure that it implements each instruction in the minimal instruction set.

Load and Store

Load and Store register operations are done in two instructions: load MAR and load or store register. The load MAR instruction places register

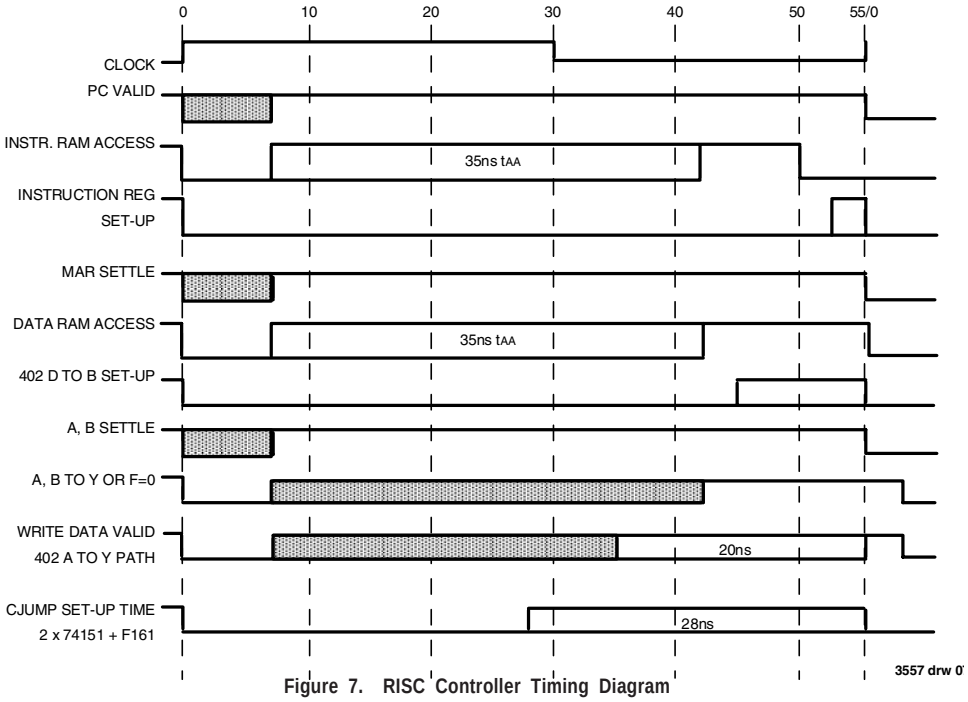


Figure 7. RISC Controller Timing Diagram

Table 2. Critical Path Timing

CONTROL PATH		DATA PATH	
PC Settle: FCT161A	6.5ns	MAR Settle: FCT161A	6.5ns
SRAM Access	35.0ns	SRAM Access	35.0ns
I reg set-up: FCT374A	2.5ns	IDT49C402A, Din Set-up	10.0ns
Total	**44.0ns**		**51.5ns**

data from the IDT49C402 or data from the immediate data field on the bus and enables MAR load. The load register instruction gates memory data into the data inputs of the IDT49C402. The store register instruction gates register data onto the bus and writes it into memory.

Move, Combine and Shift Register

Register-to-register and shift operations are performed directly by the IDT49C402 bit-slice.

Status Register Read/Write

Read and Write Status register operations select the Status Register and bus read and write, respectively.

Jump and Conditional Jump

Jump operations are done by enabling the PC to be loaded from the bus using either immediate or register data for the jump address. Conditional Jump is done by enabling a conditions select multiplexer to conditionally enable the PC load.

Jump and Save Return

The Jump and Save Return operation is performed by using the immediate data field to provide the jump address and simultaneously storing the PC in a register selected by the B field. The immediate datafield is gated to the bus, the PC is gated to the IDT49C402 data inputs and the IDT49C402 is instructed to perform a D-input-to-register-load operation.

RISC Controller Timing

The design in Figure 4 is capable of a 55ns cycle time. A timing diagram for a 55ns cycle time, assuming the 35ns dual-port SRAMs, is shown in Figure 7. The critical timing path, in this case, is the data path from the Memory Address Register (MAR) through the data port of the memory into the IDT49C402. If the dual-port SRAMs are slower than 35ns, the cycle is extended proportionately.

RISC Controller Application

The utility of the RISC controller design approach is that it allows interactive system development, debugging and diagnostic testing. It also provides the potential for high-level language support of the bit-slice design. Powerful interactive access to the RISC controller can be provided by an RS-232 interface and a FORTH language interpreter program. This allows interactive coding and testing of the system, speeding up the test-and-analyze debug cycles. This RS-232 interface can exist on a separate board external to the RISC controller, connected to the bus by a connector on the controller board. No additional hardware is required for access by the designer to the system and this access can allow direct activation and sensing of controller hardware, setting up timing loops for oscilloscope checks and on-line development of routines. If a floppy disk controller is included in the external I/O board, the RISC controller can function as a stand-alone development system in the same fashion as other stand-alone FORTH systems.

The RISC controller's ability to load programs also means that diagnostics can be loaded from the initial load EPROM . The initial load EPROM can hold both the normal control program and various test programs. The controller can load diagnostic programs from the EPROM for board and system test without requiring permanent space for them in the control memory. This allows self-diagnostics at the hardware level with minimum cost impact on the hardware.

Summary

The RISC controller uses high-speed dual-port SRAMs to blend the features of a bit-slice controller with the capabilities of a RISC computer, allowing the microinstructions of the bit-slice approach to become the instructions of a computer. This design approach provides all the speed and architectural flexibility of microcoded bit-slice designs, while allowing the use of interactive debugging methods associated with microprocessors to shorten development time.

IMPORTANT NOTICE AND DISCLAIMER

RENESAS ELECTRONICS CORPORATION AND ITS SUBSIDIARIES (“RENESAS”) PROVIDES TECHNICAL SPECIFICATIONS AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD-PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for developers who are designing with Renesas products. You are solely responsible for (1) selecting the appropriate products for your application, (2) designing, validating, and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. Renesas grants you permission to use these resources only to develop an application that uses Renesas products. Other reproduction or use of these resources is strictly prohibited. No license is granted to any other Renesas intellectual property or to any third-party intellectual property. Renesas disclaims responsibility for, and you will fully indemnify Renesas and its representatives against, any claims, damages, costs, losses, or liabilities arising from your use of these resources. Renesas' products are provided only subject to Renesas' Terms and Conditions of Sale or other applicable terms agreed to in writing. No use of any Renesas resources expands or otherwise alters any applicable warranties or warranty disclaimers for these products.

(Disclaimer Rev.1.01 Jan 2024)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact Information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit www.renesas.com/contact-us/.

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.