

An Introduction to Computer
Engineering using the Renesas
Sakura Microcontroller Board

BY JAMES M. CONRAD



An Introduction to Computer Engineering using the Renesas Sakura Microcontroller Board

BY JAMES M. CONRAD



Micrium Press
1290 Weston Road, Suite 306
Weston, FL 33326
USA

www.micrium.com

Designations used by companies to distinguish their products are often claimed as trademarks. In all instances where Micrium Press is aware of a trademark claim, the product name appears in initial capital letters, in all capital letters, or in accordance with the vendor's capitalization preference. Readers should contact the appropriate companies for more complete information on trademarks and trademark registrations. All trademarks and registered trademarks in this book are the property of their respective holders.

Copyright © 2014 by James M. Conrad except where noted otherwise. Published by Micrium Press. All rights reserved. Printed in the United States of America. No part of this publication may be reproduced or distributed in any form or by any means, or stored in a database or retrieval system, without the prior written permission of the publisher; with the exception that the program listings may be entered, stored, and executed in a computer system, but they may not be reproduced for publication.

The programs and code examples in this book are presented for instructional value. The programs and examples have been carefully tested, but are not guaranteed to any particular purpose. The publisher and content contributors do not offer any warranties and does not guarantee the accuracy, adequacy, or completeness of any information herein and is not responsible for any errors or omissions. The publisher and content contributors assume no liability for damages resulting from the use of the information in this book or for any infringement of the intellectual property rights of third parties that would result from the use of this information.

Library of Congress subject headings:

1. Embedded computer systems
2. Real-time data processing
3. Computer software—Development

For bulk orders, please contact Micrium Press at: +1 954 217 2036

ISBN: 978-1-935772-92-7

Please report errors or forward any comments and suggestions to jmconrad@uncc.edu.

Preface

This book is the result of a long relationship the author has enjoyed with Renesas Electronics America, Inc. (and one of its predecessors, Mitsubishi Electronics). I originally worked with this company because of their commitment to providing a low-cost evaluation board and free development software that students could purchase and use in classes and senior design projects. Over the years the boards have remained as affordable (and popular) as ever, and the software development tools available have added more functionality while still available for free to our students.

I have been teaching embedded systems courses for over fourteen years (and working in the field even longer). I had not been able to find a book suitable for using in an Introduction to Computer Engineering course that would lend itself to the theoretical and applied nature of the discipline and embedded systems design. When Renesas released the GR-SAKURA board, I knew I have the perfect platform to use in the course. This book was developed to augment the hands-on exercises we use. This book also has a radical feature not seen in many books currently on the market (if any). It is freely available for download. It is also available for purchase in hardcopy form for a modest price.

This book would not have been possible had it not been for the assistance of numerous people. Several students and educators contributed to and extensively tested some of the chapters, including: Yevgeny Fridlyand (2, 3, 4), Adam Harris (1, 2, 3), Anthony Harris (3), Onkar Raut (2, 4) Suganya Jebasingh (2, 4), and Steven Erdmanczyk (4) . Thanks go to the publisher, Linda Foegen, and especially June Harris, Rob Dautel and Todd DeBoer of Renesas for their help in getting this book produced and published (and for their patience!). Many, many thanks go to the reviewers who offered valuable suggestions to make this book better, especially David Brown and students from my UNC Charlotte Introduction to Engineering and Embedded Systems courses.

I would like to personally thank my parents, the Conrads, and my in-laws, the Warrens, for their continued assistance and guidance through the years while I worked on books. Also, I would especially like to thank my children, Jay, Mary Beth, and Caroline, and my wife Stephanie for their understanding when I needed to spend more time on the book than I spent with them.

James M. Conrad, March 2014

Contents

| | |
|---|----------|
| Preface | iii |
| Foreword | v |
| CHAPTER 1 | |
| <hr/> | |
| Introduction to Embedded Systems | 1 |
| 1.1 Learning Objectives | 1 |
| 1.2 Concepts | 1 |
| 1.2.1 Economics and Microcontrollers | 1 |
| 1.2.2 Embedded Networks | 2 |
| 1.3 Typical Benefits of Embedded Systems | 2 |
| 1.3.1 Greater Performance and Efficiency | 3 |
| 1.3.2 Lower Costs | 3 |
| 1.3.3 More Features | 4 |
| 1.3.4 Better Dependability | 4 |
| 1.4 Embedded System Functions | 4 |
| 1.5 Attributes of Embedded Systems | 5 |
| 1.6 Constraints on Embedded Systems | 6 |
| 1.7 Developing Embedded Systems | 6 |
| 1.7.1 Product Development | 7 |
| 1.7.2 Designing and Manufacturing Embedded Systems | 8 |
| 1.7.3 The Role of a Computer Engineer | 9 |
| 1.8 An Example of an Embedded System: The Renesas Sakura Board | 10 |
| 1.9 Summary of Book Contents | 10 |
| 1.10 Recap | 10 |
| 1.11 References | 11 |

Introduction to Embedded Systems

1.1 LEARNING OBJECTIVES

In this chapter the reader will learn:

- What an embedded system is
- Why to embed a computer
- What functions and attributes embedded systems need to provide
- What constraints embedded systems have

1.2 CONCEPTS

An embedded system is an application-specific computer system which is built into a larger system or device. Using a computer system rather than other control methods (such as non-programmable logic circuits, electro-mechanical controls, and hydraulic controls) offers many benefits such as sophisticated control, precise timing, low unit cost, low development cost, high flexibility, small size, and low weight. These basic characteristics can be used to improve the overall system or device in various ways:

- Improved performance
- More functions and features
- Reduced cost
- Increased dependability

Because of these benefits, billions of microcontrollers are sold each year to create embedded systems for a wide range of products.

1.2.1 Economics and Microcontrollers

Microcontrollers are remarkably inexpensive yet offer tremendous performance. The microprocessor for a personal computer may cost \$100 or more, while microcontrollers typically cost far less, starting at under \$0.25. Why is this so?

2.3 MICROCONTROLLER BASICS

2.3.1 Bits and Bytes

The basic concept of an embedded system is electricity. If we ignore the underlying voltage value and just consider the maximum voltage of the system, it is easy to recognize two conditions:

- presence of the maximum voltage of the system—we'll call this state "1"
- absence of a voltage of the system (most often 0 (zero) volts)—we'll call this state "0"

This basic unit of information is the *binary digit*, or *bit*. Values with more than two states require multiple bits. Therefore a collection of two bits has four possible states: 00, 01, 10, and 11. A collection of eight bits is called a *byte*. Often we group bits together to represent them in a larger number representation, called hexadecimal. A grouping of four bits is represented by one hexadecimal digit, usually preceded by an 'x,' as represented in Table 2.1. As an example, the binary number 1010 is xA in hexadecimal and 10 in decimal. Binary number 01011100 is hexadecimal x5C.

TABLE 2.1 Hexadecimal Representation

| BINARY | HEXADECIMAL | DECIMAL | BINARY | HEXADECIMAL | DECIMAL |
|--------|-------------|---------|--------|-------------|---------|
| 0000 | x0 | 0 | 1000 | x8 | 8 |
| 0001 | x1 | 1 | 1001 | x9 | 9 |
| 0010 | x2 | 2 | 1010 | xA | 10 |
| 0011 | x3 | 3 | 1011 | xB | 11 |
| 0100 | x4 | 4 | 1100 | xC | 12 |
| 0101 | x5 | 5 | 1101 | xD | 13 |
| 0110 | x6 | 6 | 1110 | xE | 14 |
| 0111 | x7 | 7 | 1111 | xF | 15 |

These values are moved around inside the microcontroller and stored in memory locations called registers. Each register has a unique location which will be addressed. These memory locations are in addition to larger stores of useable memory.

union of variables within that structure. An example of how the PDR is defined inside a port *structure* as follows:

```
1. struct st_port4 {
2.     union {
3.         unsigned char BYTE;
4.         struct {
5.             unsigned char B0:1;
6.             unsigned char B1:1;
7.             unsigned char B2:1;
8.             unsigned char B3:1;
9.             unsigned char B4:1;
10.            unsigned char B5:1;
11.            unsigned char B6:1;
12.            unsigned char B7:1;
13.        } BIT;
14.    } PDR;
15. }
```

Line 1 shows that port4 has been defined as a *structure*. Lines 2 to 14 suggest that the Port Direction Register (PDR) has been defined as a *union* with the variable BYTE and a structure called BIT. This organization helps in easy access of the bits of the PDR. Unsigned char B n :1 (n : 0 to 7) indicates that the character variable is assigned one bit.

To select a particular pin as the input pin, the corresponding bit of the PDR has to be set to '0'; and to select a pin as output, the corresponding bit of the PDR has to be set to '1.' The general syntax to set a bit of the PDR is PORT x .PDR.BIT.B n (x = 0 to 5, A to G, J; and n = 0 to 7) since ports are defined as *structures*, hence accessing structure *members* is done in this way. To configure multiple pins at the same time, the *char* variable BYTE can be used. All pins are configured as inputs at reset, by default.

Set Switch 1 (Port A bit 7) as Input

```
1. PORTA.PDR.BIT.B7 = 0;
```

When a pin is selected as an input from a peripheral, the Input Buffer Control Register (ICR) has to be enabled. The ICR will be explained a little later. Selecting a pin as an output involves setting the Data Register (DR) and the Port Direction Register (PDR).

Port Output Data Register (PODR)

The Port Output Data Register (PODR) is also defined as a *union* of variables inside the port *structure*, in the 'iodefine_gcc63n.h' file. It is presented just like the PDR. Unsigned char: 1 is used to represent reserved pins.

| | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 |
|--------------------|----|----|----|----|----|----|----|----|
| Value after reset: | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Figure 2.6 Port Output Data Register [1], page 662.

The syntax to access the bits of the Data Registers (DR) is `PORTx.PIDR.BIT.Bn` ($x = 0$ to 9, A to G, J; and $n = 0$ to 7) for those port pins configured as inputs and `PORTx.PODR.BIT.Bn` for those port pins configured as outputs. To select a pin as an output pin, first set the Port Output Data Register (PODR) to a known value, preferably 0, so that changes in the output can be easily observed. The *char* variable `BYTE` can be used to set multiple pins as output at the same time.

Set LED0 (Port A bit 0) as Output

1. `PORTA.PDR.BIT.B0 = 1;`
2. `PORTA.PODR.BIT.B0 = 0;`

Line 1 sets LED0 as an output and line 2 switches on the LED.

Sets LEDs 1, 2, 3, and 4 (Port A bit 0, 1, 2, and 3) as Outputs

1. `PORTA.PDR.BYTE = 0x47;`
2. `PORTA.PODR.BYTE = 0xB8;`

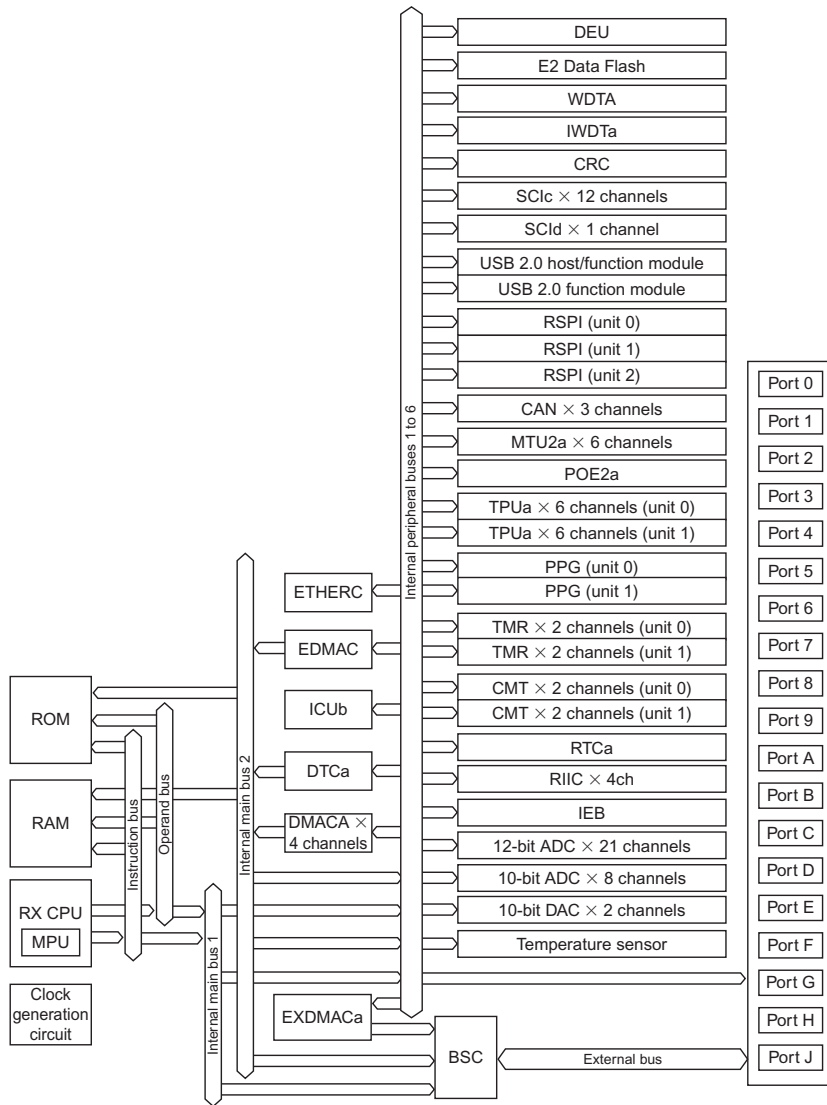
Line 1 sets LED1, 2, 3, and 4 as outputs and line 2 switches on the LEDs.

Port Input Data Register (PIDR)

The Port Input Data Register is also defined as a union of variables inside the port structure in the `'iodefine_gcc63n.h'` file. `PORTx.PIDR.BIT.Bn` ($x = 0$ to 9, A to G, J; and $n = 0$ to 7) is used to read the state of a pin and the state is stored in the Port Input Data Register regardless of the value in the Port Mode Register (PMR). This register also has some reserved bits. These bits are read as 1 and cannot be modified.

| | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
|--------------------|----|----|----|----|----|----|----|----|
| Value after reset: | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Figure 2.7 Port Input Data Register [1], Page 663.



- | | | | |
|------------|---|-------|------------------------------------|
| ETHERC | :Ethernet controller | RSP1 | :Serial peripheral interface |
| EDMAC | :DMA controller for Ethernet controller | CAN | :CAN module |
| ICUb | :Interrupt controller | MTU2a | :Multi-function timer pulse unit 2 |
| DTCa | :Data transfer controller | POE2a | :Port output enable 2 |
| DMACA | :DMA controller | TPUa | :16-bit timer pulse unit |
| EXDMACa | :EXDMA controller | PPG | :Programmable pulse generator |
| BSC | :Bus controller | TMR | :8-bit timer |
| WDTA | :Watchdog timer | CMT | :Compare match timer |
| IWDTa | :Independent watchdog timer | RTCa | :Realtime clock |
| CRC | :CRC (cyclic redundancy check) calculator | RIIC | :I ² C bus interface |
| SCIC, SCID | :Serial communications interface | IEB | :IEBus controller |
| MPU | :Memory protection unit | DEU | :Data encryption unit |

Figure 3.4 Block diagram [1], page 66.

3.4 SAKURA EXAMPLE PROJECT

Let's go through the sample project to get better acclimated with the development tools for the GR-SAKURA board. From the SAKURA board website, click on the "Try Guest Login" link.

When you first login, a window will pop-up asking you to create a project. You will need to select a "template" and a project name as seen below. Let's use the name "GR_SAKURA_Lab1" for this project.

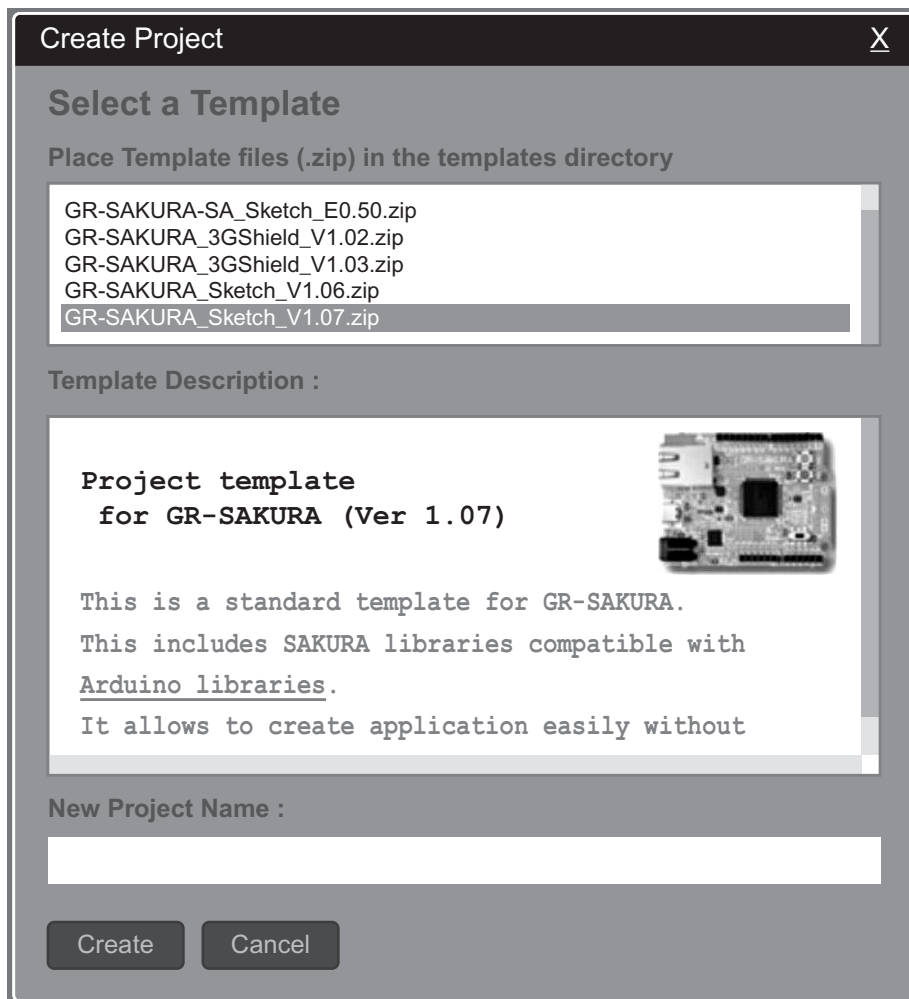


Figure 3.9 Create a project from a template.

Calculate how many rotations of the wheels we will need to travel one meter. To do this we will need to compute the circumference of the wheel then divide the forward distance by the circumference.

$$2\pi r = d\pi = 2.5 \text{ in} \times \pi \approx 7.854 \text{ in}$$

$$\frac{39.37 \text{ in}}{7.854 \text{ in/rotation}} \approx 5.012 \text{ rotations}$$

Now that we know how many rotations of the wheel we will need, we can compute the total rotations of the motor by multiplying the wheel rotations by the gear ratio.

$$5.012 \text{ wheel rotations} \times \frac{53 \text{ motor rotations}}{1 \text{ wheel rotation}} \approx 265.636 \text{ motor rotations}$$

Lastly, let's identify the total time that we will need to enable the motors in forward motion by taking the total motor rotations and dividing it by the rated RPM and converting to milliseconds.

$$\frac{265.636 \text{ motor rotations}}{6700 \text{ rotations/min}} \approx 0.039647 \text{ min}$$

$$0.039647 \text{ min} \times \frac{60 \text{ sec}}{1 \text{ min}} \times \frac{1000 \text{ ms}}{1 \text{ sec}} \approx 2378.82 \text{ ms}$$

We need to calculate the total distance that the wheels will need to rotate in opposite directions to make a 90 degree turn. First, calculate the distance from one wheel to the other. This can be achieved by simply measuring the platform from the center of one wheel to the other. To achieve a 90 degree turn, each wheel will need to travel $\frac{1}{4}$ of the total circumference in opposite directions.

Figure 3.39 shows a diagram of how we would expect the vehicle to move about its axis when the wheels are turning in opposite directions.

$$\frac{6.61685 \text{ in}}{7.854 \text{ in/rotation}} \approx 0.8425 \text{ wheel rotations}$$

$$0.8425 \text{ wheel rotations} \times \frac{53 \text{ motor rotations}}{1 \text{ wheel rotation}} \approx 44.65 \text{ motor rotations}$$

$$\frac{44.65 \text{ motor rotations}}{6700 \text{ motor rotations/min}} \approx 0.00666418 \text{ min}$$

$$0.00666418 \text{ min} \times \frac{60 \text{ sec}}{1 \text{ min}} \times \frac{1000 \text{ ms}}{1 \text{ sec}} \approx 399.85 \text{ ms}$$

8. Update the direction of the left wheel to backward, keep the right wheel direction forward
9. Enable the motors for 0.399 seconds to make a 90 degree turn
10. Disable the motors for 0.5 seconds (keep H-bridge from shorting)
11. Update the direction of the left wheel to forward, keep the right wheel direction forward
12. Repeat steps 6–11 three more times
13. Disable the motors

Once we are done with our algorithm, we can begin to code. The below code example satisfies the algorithm that we have created. But does it satisfy the customer requirements? Build the project using the code below and download it to the Sakura board. Make sure to connect the connectors to the pins defined in the code.

```
1. /*GR-SAKURA Sketch Template Version: V1.08*/
2. #include <rxduino.h>
3.
4. #define directionA 2 //Port 2 pin 2 - motor A direction
5. #define enableA 3 //Port 2 pin 3 - motor A on/off
6. #define directionB 4 //Port 2 pin 4 - motor B direction
7. #define enableB 5 //Port 2 pin 5 - motor B on/off
8. #define Aforward 1
9. #define Abackward 0
10. #define Bforward 1
11. #define Bbackward 0
12. #define FORWARDTIME 2380 //forward delay in milliseconds
13. #define TURNTIME 400 //turn delay in milliseconds
14. #define PAUSE 500 //Pause between switching directions
15. #define ON 1
16. #define OFF 0
17. int i; //counter variables
18. //*****
19. //setup input/output pins
20. //
21. //*****
22. void setup() {
23.     pinMode(directionA, OUTPUT);
24.     pinMode(enableA, OUTPUT);
25.     pinMode(directionB, OUTPUT);
26.     pinMode(enableB, OUTPUT);
```

```
66.     }  
67. } //end program
```

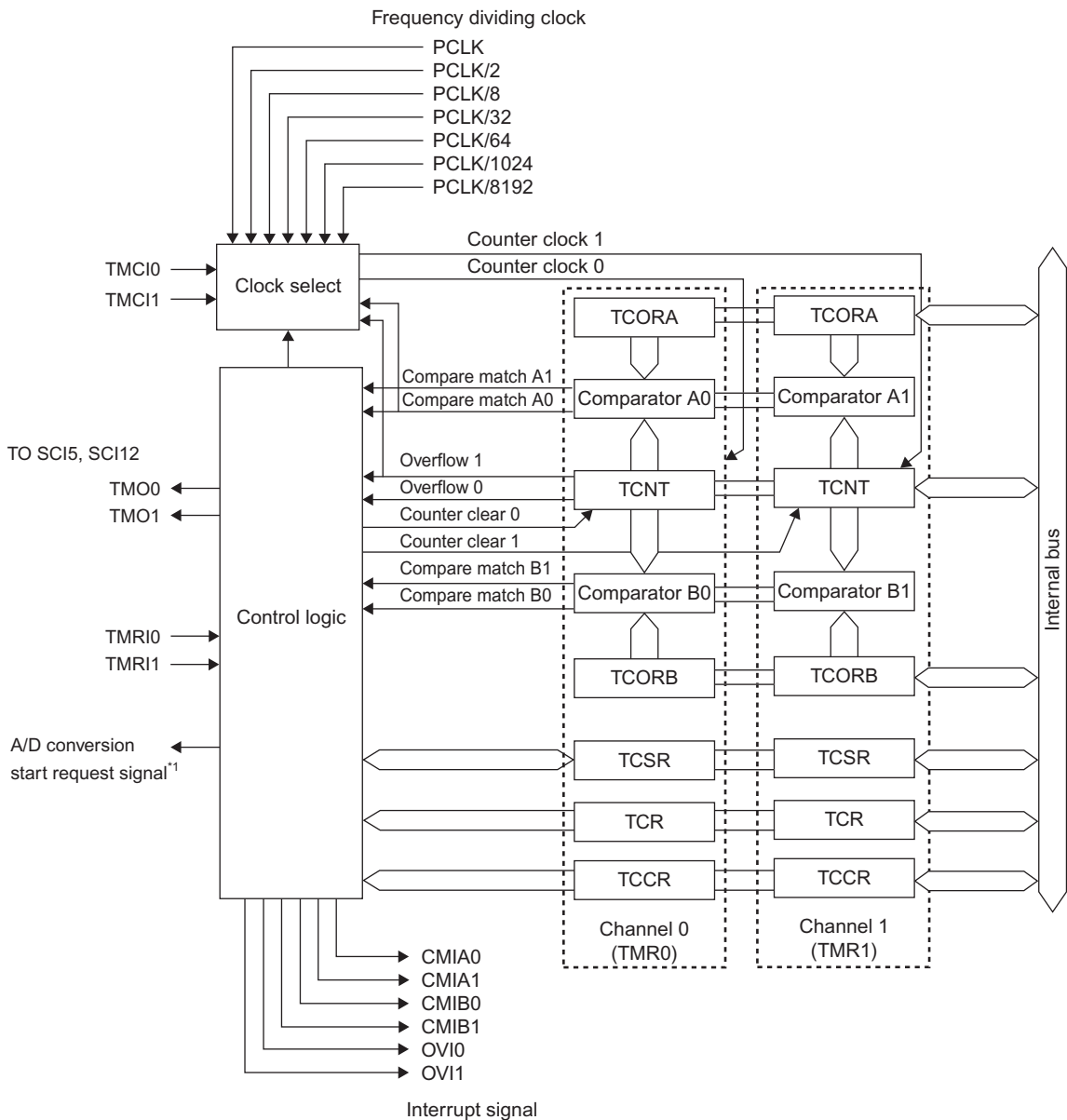
Did the robot complete a 1 x 1 meter square? If the answer is yes then consider yourself lucky. If no, then it should not be a surprise. There are many variables that the above code does not consider. We use the rated speed for the motors, but it is more than likely that the RPM varies slightly from motor to motor. In fact, it could possibly be off by as much as 500 rpm. We also did not accommodate for or considered the weight of the robot, otherwise known as the payload. The more weight on the motors the slower they will rotate. How about the terrain? It would make a substantial difference in the rpm of the motors from riding on a smooth wooden surface or on a grassy field. How about slippage? It is quite possible for one of the wheels to slip slightly or even rotate faster than the other. To accommodate for our environment we will require feedback mechanisms, counters, and pulse width modulators. All this we will learn in Chapter 4, where we will attempt to utilize our knowledge to achieve the functionality dictated by our requirements.

3.9 RECAP

The GR-Sakura is one of the Gadget Renesas board series. It is based on RX63N series 32-bit MCU. The MCU has on-chip flash memory and enhanced communication functions, including an Ethernet controller and USB 2.0 Host/Function. The on-chip flash memory of RX63N is programmable by USB mass storage mode, and the on-chip flash memory is visible as a drive on your PC. This chapter presented the tools and processes to use the Sakura embedded board. The board is easy to program and use for many applications, including sensing applications and robotics.

3.10 REFERENCES

- [1] Renesas Electronics, Inc. (February, 2013). *RX63N Group, RX631 Group User's Manual: Hardware, Rev. I.60*.
- [2] Conrad, James M. (2013). *Embedded Systems: An Introduction using the RX63N Microcontroller*. Micrium Press.
- [3] Digilent Inc. (August, 2012). *Motor Robot Kit (MRK) Reference Manual, Rev.*
- [4] Digilent Inc. (February 28, 2012). *Digilent PmodHBS™ 2A H-Bridge Reference Manual, Circuit Rev D, Document Rev.*
- [5] Shayang Ye. (April, 2010). *DC Carbon-brush motors, IG-22 Geared Motor Series: IG-22GP Type 01 & 02*.



| | |
|---------------------------------|--------------------------------------|
| TCORA: Time constant register A | TCSR: Timer control/status register |
| TCNT: Timer counter | TCR: Timer control register |
| TCORB: Time constant register B | TCCR: Timer counter control register |

Note: * For the corresponding A/D converter channels, see section 40, 12-Bit A/D Converter (S12ADa), and section 41, 10-Bit A/D Converter (ADb).

Figure 4.2 Block diagram of TMR Unit 0 [1], page 1014.

4.2.1 Setting Up a Timer for Counting Events

Timer Count Register

The TCNT (Timer Counter) register holds the current timer value at any time after the timer has been configured. Whenever you want to know the value of the timer or counter you will read the value in this register. Also, when not currently operating the timer, you can load a value into this register and the timer will begin counting from that value when restarted. Note that in the 16-bit mode TMR0.TCNT and TMR1.TCNT (TMR2.TCNT and TMR3.TCNT as well) cascade into one 16-bit register. This holds true for the timer constant registers as well.

Address(es): TMR0.TCNT 0008 8208h, TMR1.TCNT 0008 8209h
TMR2.TCNT 0008 8218h, TMR3.TCNT 0008 8219h

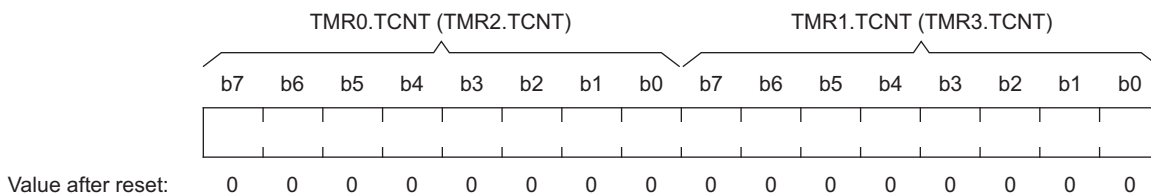


Figure 4.3 Timer Counter (TCNT) Register [1], page 1017.

The TCNT register is where the count is held. After the timer is started this register will increment every time a count is detected. If you want to know the current count, you can read this register. If the timer is stopped you can write a value to this register and it will begin counting from the written value when re-started.

Timer Counter Control Register

The Timer Counter Control Register (TCCR) controls where the timers count source comes from. Dependent on what value is set here, it will be determined if the count comes from the internal peripheral clock, a pre-scaled peripheral clock, an external count source, or from another timer overflowing. This register also enables the timer's interrupts on the peripheral level.

Address(es): TMR0.TCCR 0008 820Ah, TMR1.TCCR 0008 820Bh
TMR2.TCCR 0008 821Ah, TMR3.TCCR 0008 821Bh

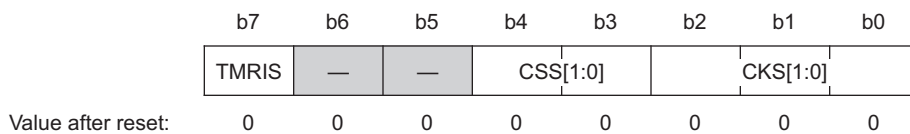


Figure 4.4 Timer Counter Control Register [1], page 1019.

| BIT | SYMBOL | BIT NAME | DESCRIPTION | R/W |
|----------|----------|--|--|-----|
| b2 to b0 | CKS[2:0] | Clock Select* | See table below. | R/W |
| b4, b3 | CSS[1:0] | Clock Source Select | See table below. | R/W |
| b6, b5 | — | (Reserved) | These bits are always read as 0. The write value should always be 0. | R/W |
| b7 | TMRIS | Timer Reset Detection Condition Select | 0: Cleared at rising edge of the external reset | R/W |
| | | | 1: Cleared when the external reset is high | |

Note: * To use an external clock, set the Pn.PDR.Bi bit for the corresponding pin to "0" and the PORTn.PMR.Bi bit to "1". For details, see [1] section 21, I/O Ports.

Figure 4.5 TCCR (Timer Counter Control Register) description [1], page 1019.

| CHANNEL | TCCR REGISTER | | | | | DESCRIPTION | | |
|----------------|---------------|---------------------|----------|----|----|--|---|---|
| | CSS[1:0] | | CKS[2:0] | | | | | |
| | B4 | B3 | B2 | B1 | B0 | | | |
| TMR0 (TMR2) | 0 | 0 | — | 0 | 0 | Clock input prohibited. | | |
| | | | | | 1 | Uses external clock. Counts at rising edge* ¹ . | | |
| | | | | | 0 | Uses external clock. Counts at falling edge* ¹ . | | |
| | | | | | 1 | Uses external clock. Counts at both rising and falling edges* ¹ . | | |
| | 0 | 1 | 0 | 0 | 0 | Uses internal clock. Counts at PCLK. | | |
| | | | | | 1 | Uses internal clock. Counts at PCLK/2. | | |
| | | | | | 1 | 0 | Uses internal clock. Counts at PCLK/8. | |
| | | | | | | 1 | Uses internal clock. Counts at PCLK/32. | |
| | | | | | 1 | 0 | 0 | Uses internal clock. Counts at PCLK/64. |
| | | | | | | | 1 | Uses internal clock. Counts at PCLK/1024. |
| | | | | | | | 0 | Uses internal clock. Counts at PCLK/8192. |
| | | | | | 1 | 1 | — | — |
| | 1 | Setting prohibited. | | | | | | |
| | 1 | 1 | — | — | — | Counts at TMR1.TCNT (TMR3.TCNT) overflow signal* ² . | | |

Figure 4.6 Clock input to TCNT and count condition [1], page 1020.

| CHANNEL | TCCR REGISTER | | | | | DESCRIPTION | | |
|----------------|---------------|----|----------|----|----|--|---|---|
| | CSS[1:0] | | CKS[2:0] | | | | | |
| | B4 | B3 | B2 | B1 | B0 | | | |
| TMR1 (TMR3) | 0 | 0 | — | 0 | 0 | Clock input prohibited. | | |
| | | | | | 1 | Uses external clock. Counts at rising edge* ¹ . | | |
| | | | | | 1 | 0 | Uses external clock. Counts at falling edge* ¹ . | |
| | | | | | 1 | Uses external clock. Counts at both rising and falling edges* ¹ . | | |
| | 0 | 1 | 0 | 0 | 0 | 0 | Uses internal clock. Counts at PCLK. | |
| | | | | | 1 | Uses internal clock. Counts at PCLK/2. | | |
| | | | | | 1 | 0 | Uses internal clock. Counts at PCLK/8. | |
| | | | | | 1 | Uses internal clock. Counts at PCLK/32. | | |
| | | | | 1 | 0 | 0 | 0 | Uses internal clock. Counts at PCLK/64. |
| | | | | | | 1 | Uses internal clock. Counts at PCLK/1024. | |
| | | | | | | 1 | 0 | Uses internal clock. Counts at PCLK/8192. |
| | | | | | | 1 | Clock input prohibited. | |
| | 1 | 0 | — | — | — | Setting prohibited. | | |
| | 1 | 1 | — | — | — | Counts at TMR0.TCNT (TMR2.TCNT) overflow signal* ² . | | |

Notes:

1. To use an external clock, set the PORTn.PDR.Bi bit for the corresponding pin to “0” and the PORTn.OPMR.Bi bit to “1”. For details, see [1] section 21, I/O Ports.
2. If the clock input of TMR0 (TMR2) is the overflow signal of the TMR1.TCNT (TMR3.TCNT) counter and that of TMR1 (TMR3) is the compare match signal of the TMR0.TCNT (TMR2.TCNT) counter, no incrementing clock is generated. Do not use this setting.

Figure 4.6 Clock input to TCNT and count condition [1], page 1020.—Continued.

Time Constant Register

The TCORA (Time Constant Register A) and TCORB (Time Constant Register B) are used to store constants to compare against the TCNT register. Every time the TCNT increments it is constantly being compared against either of these registers. When TCNT matches either of these registers, a compare match event occurs. Compare match events have many uses depending on what mode we are using the timer in.

Address(es): TMR0.TCORA 0008 8204h, TMR1.TCORA 0008 8205h
 TMR2.TCORA 0008 8214h, TMR3.TCORA 0008 8215h

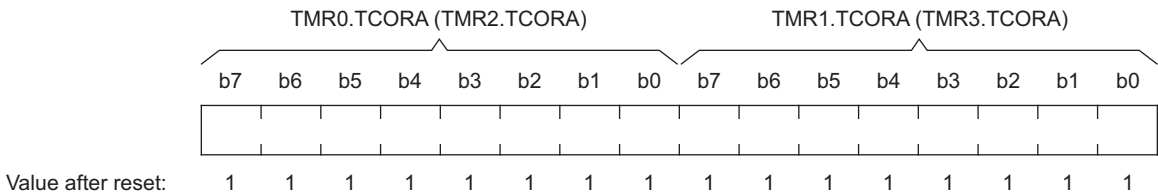
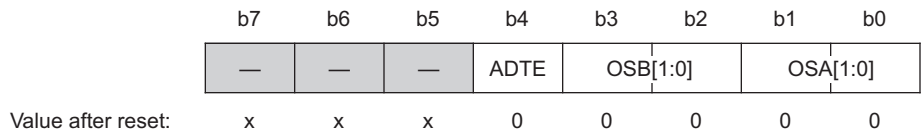


Figure 4.7 Time Constant Register A[1], page 1017.

Timer Control/Status Register

The TCSR (Timer Control/Status Register) register controls compare match output. Each timer has an output port assigned to it which is controlled via compare match events. This is one of many uses of the compare match events. When a compare match event occurs this register can set the output of the timer’s port to 1 or 0, or toggle it. This register is used when we want the timer to control a pulse output.

Address(es): TMR0.TCSR 0008 8202h, TMR2.TCSR 0008 8212h



Address(es): TMR0.TCSR 0008 8203h, TMR3.TCSR 0008 8213h

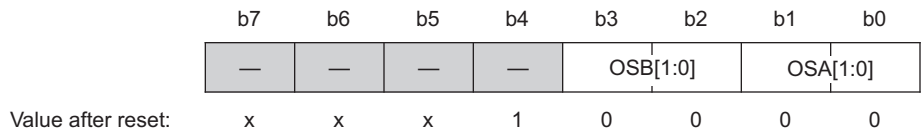


Figure 4.8 Timer Control/Status Register [1], page 1021.

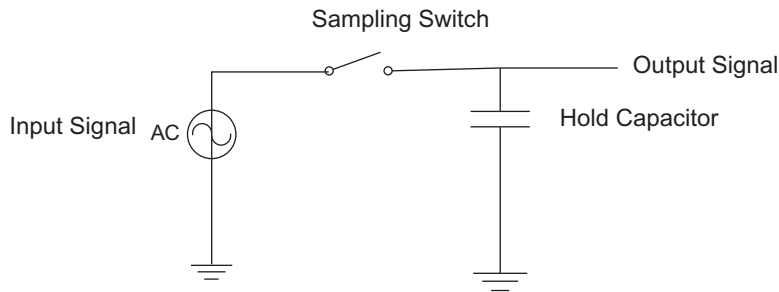


Figure 4.18 Sample and Hold Circuit.

Signal from the sample and hold circuit is then given to the comparator. The comparator compares the input signal with a reference signal and gives the digital output. The reference voltage of an ADC is the maximum analog voltage that can be converted by an ADC.

The digital value for a particular analog value can be found mathematically. This can be useful as a guide to see if the ADC output obtained is correct.

If V_{in} is the sampled input voltage, V_{+ref} is the upper end of input voltage range, V_{-ref} is the lower end of input voltage range, and N is the number of bits of resolution in ADC, the digital output (n) can be found using the following formula:

$$n = \left[\frac{(V_{in} - V_{-ref})(2^N - 1)}{V_{+ref} - V_{-ref}} + \frac{1}{2} \right] \text{int}$$

$$n = \left[\frac{(V_{in})(2^N - 1)}{V_{+ref}} + \frac{1}{2} \right] \text{int} \text{ (if } V_{-ref} = 0 \text{)}$$

Let us assume that the analog voltage to be calculated is 2.7 V and the 12-bit ADC has to be used. The digital value will be:

$$n = \left[\frac{(V_{in})(2^N - 1)}{V_{+ref}} + \frac{1}{2} \right] \text{int} \text{ (Since } V_{-ref} = 0 \text{)} = \left[\frac{(2.7)(2^{12} - 1)}{3.3} + \frac{1}{2} \right] \text{int} = \left[\frac{(2.7)(4095)}{3.3} + \frac{1}{2} \right] \text{int}$$

$$n = 3352_{10}$$

The GR-SAKURA has one 10-bit A/D converter unit and one 12-bit A/D converter unit. However, the GR-SAKURA is designed specifically to use as little real-estate as possible, hence there are not enough designated pins on the board to support all of the RX63N MCU functionality at the same time. There are six designated A/D pins (AN0 to AN5) that can only be used for the 12-bit A/D.

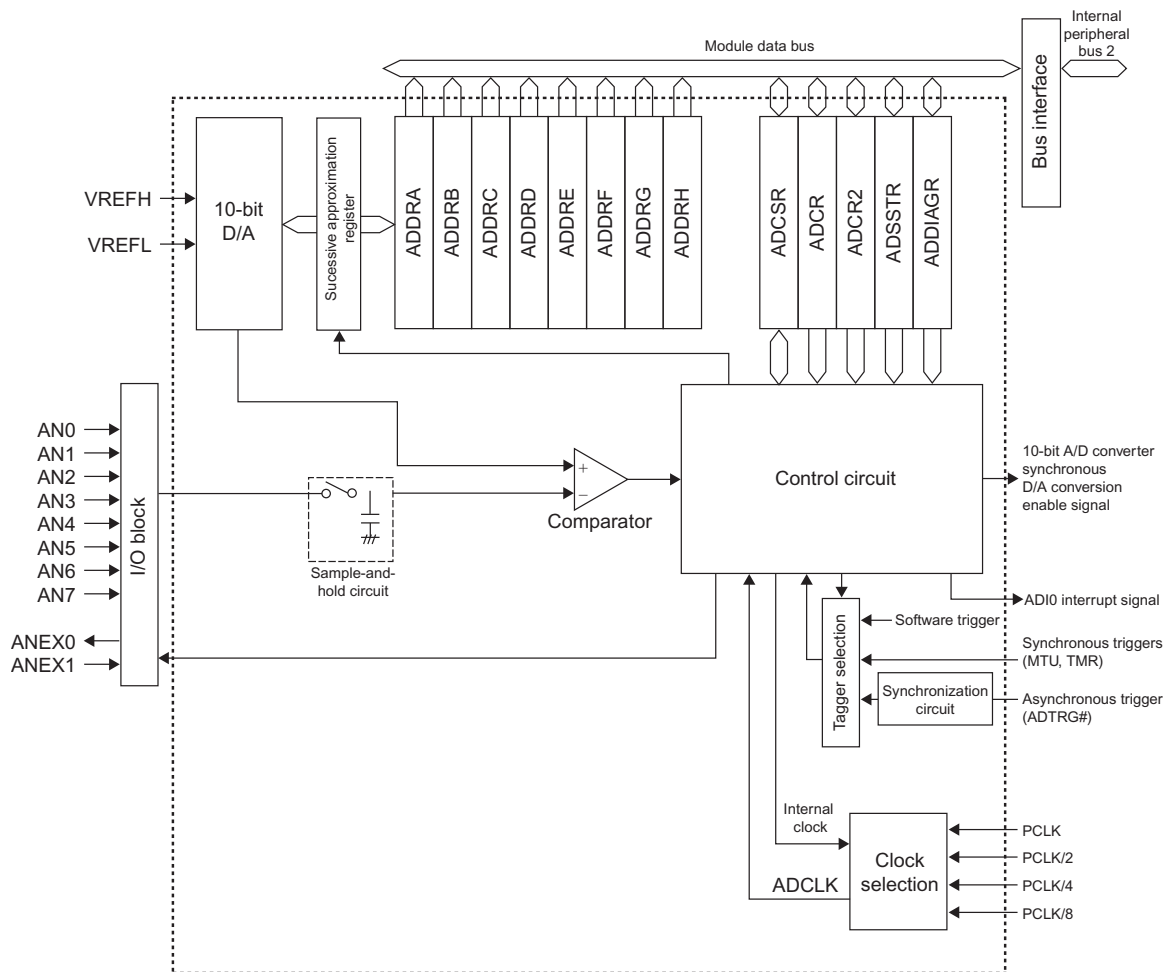
TABLE 4.3 GR-Sakura 12-bit A/D Converter Port Map [8].

| CN15 | PIN NUMBER 100 PIN LQFP | I/O PORT | BUS EXDMAC | TIMER (MTU, TPU, TMR, PPG, RTC, POE) | COMMUNICATIONS (ETHERC, SC1c, SC1d, RSPI, RIIC, CAN, IEB, USB) | INTERRUPT | S12AD, AD, DA |
|------|-------------------------------|----------|---------------|--|--|-----------|---------------------|
| AD0 | 95 | P40 | | | | IRQ8-DS | AN000 |
| AD1 | 93 | P41 | | | | IRQ9-DS | AN001 |
| AD2 | 92 | P42 | | | | IRQ10-DS | AN002 |
| AD3 | 91 | P43 | | | | IRQ11-DS | AN003 |
| AD4 | 90 | P44 | | | | IRQ12-DS | AN004 |
| AD5 | 89 | P45 | | | | IRQ13-DS | AN005 |

The 10-bit ADC cannot be utilize with the defined A/D pins AN001-AN005. However, the GR-SAKURA does provide access to the 10-bit ADC using the pins shown in Table 4.4A.

TABLE 4.4A GR-Sakura 10-bit A/D Converter Port Map[8].

| CN15 | PIN NUMBER 100 PIN LQFP | I/O PORT | BUS EXDMAC | TIMER (MTU, TPU, TMR, PPG, RTC, POE) | COMMUNICATIONS (ETHERC, SC1c, SC1d, RSPI, RIIC, CAN, IEB, USB) | INTERRUPT | S12AD, AD, DA |
|------|-------------------------------|----------|---------------|--|--|-----------|---------------------|
| IO44 | 78 | PE0 | D8[A8/D8] | | SCK12/ SSLB1 | | ANEX0 |
| IO45 | 77 | PE1 | D9[A9/D9] | MTIOC4C/ PO18 | TXD12/ SMOSI12/ SSDA12/ TXDX12/ SIOX12/ SSLB2/ RSPCKB | | ANEX1 |
| IO46 | 76 | PE2 | D10[A10/D10] | MTIOC4A/ PO23 | RXD12/ SMISO12/ SSCL12/ RXDX12/ SSLB3/MOSIB | IRQ7-DS | AN0 |
| IO47 | 75 | PE3 | D11[A11/D11] | MTIOC4B/ PO26/ POE8# | CTS12#/ RT S12#/ SS12#/ MISOB/ ET_ERXD3 | | AN1 |
| IO48 | 74 | PE4 | D12[A12/D12] | MTIOC4D/ MTIOC1A/ PO28 | SSLB0/ ET_ERXD2 | | AN2 |
| IO49 | 73 | PE5 | D13[A13/D13] | MTIOC4C/ MTIOC2B | RSPCKB/ ET_RX_CLK/ REF50CK | IRQ5 | AN3 |
| IO50 | 72 | PE6 | D14[A14/D14] | | MOSIB | IRQ6 | AN4 |
| IO51 | 71 | PE7 | D15[A15/D15] | | MOSIB | IRQ7 | AN5 |
| IO42 | 80 | PD6 | D6[A6/D6] | MTIC5V/POE1# | | IRQ6 | AN6 |
| IO43 | 79 | PD7 | D7[A7/D7] | MTIC5U/POE0# | | IRQ7 | AN7 |



| | |
|----------------------------|---------------------------------------|
| ADDRA: A/D data register A | ADCSR: A/D control/status register |
| ADDRB: A/D data register B | ADCR: A/D control register |
| ADDRC: A/D data register C | ADCR2: A/D control register 2 |
| ADDRD: A/D data register D | ADSSTR: A/D sampling state register |
| ADDRE: A/D data register E | ADDIAGR: A/D self-diagnostic register |
| ADDRF: A/D data register F | |
| ADDRG: A/D data register G | |
| ADDRH: A/D data register H | |

Figure 4.20 Block Diagram of the 10-bit A/D Converter [1], page 1684.

TABLE 4.5 List of 10-bit A/D Converter Registers [2], page 77.

| ADDRESS | MODULE SYMBOL | REGISTER NAME | REGISTER SYMBOL | NUMBER OF BITS | ACCESS SIZE | NUMBER OF ACCESS STATES | | RELATED FUNCTION |
|------------|---------------|------------------------------|-----------------|----------------|-------------|-------------------------|---------------|------------------|
| | | | | | | ICLK \geq PCLK | ICLK $<$ PCLK | |
| 0008 9800h | AD | A/D data register A | ADDRA | 16 | 16 | 2, 3 PCLKB | 2 ICLK | ADb |
| 0008 9802h | AD | A/D data register B | ADDRB | 16 | 16 | 2, 3 PCLKB | 2 ICLK | |
| 0008 9804h | AD | A/D data register C | ADDRC | 16 | 16 | 2, 3 PCLKB | 2 ICLK | |
| 0008 9806h | AD | A/D data register D | ADDRD | 16 | 16 | 2, 3 PCLKB | 2 ICLK | |
| 0008 9808h | AD | A/D data register E | ADDRE | 16 | 16 | 2, 3 PCLKB | 2 ICLK | |
| 0008 980Ah | AD | A/D data register F | ADDRF | 16 | 16 | 2, 3 PCLKB | 2 ICLK | |
| 0008 980Ch | AD | A/D data register G | ADDRG | 16 | 16 | 2, 3 PCLKB | 2 ICLK | |
| 0008 980Eh | AD | A/D data register H | ADDRH | 16 | 16 | 2, 3 PCLKB | 2 ICLK | |
| 0008 9810h | AD | A/D control/status register | ADCSR | 8 | 8 | 2, 3 PCLKB | 2 ICLK | |
| 0008 9811h | AD | A/D control register | ADCR | 8 | 8 | 2, 3 PCLKB | 2 ICLK | |
| 0008 9812h | AD | A/D control register 2 | ADCR2 | 8 | 8 | 2, 3 PCLKB | 2 ICLK | |
| 0008 9813h | AD | A/D sampling state register | ADSSTR | 8 | 8 | 2, 3 PCLKB | 2 ICLK | |
| 0008 981Fh | AD | A/D self-diagnostic register | ADDIAGR | 8 | 8 | 2, 3 PCLKB | 2 ICLK | |

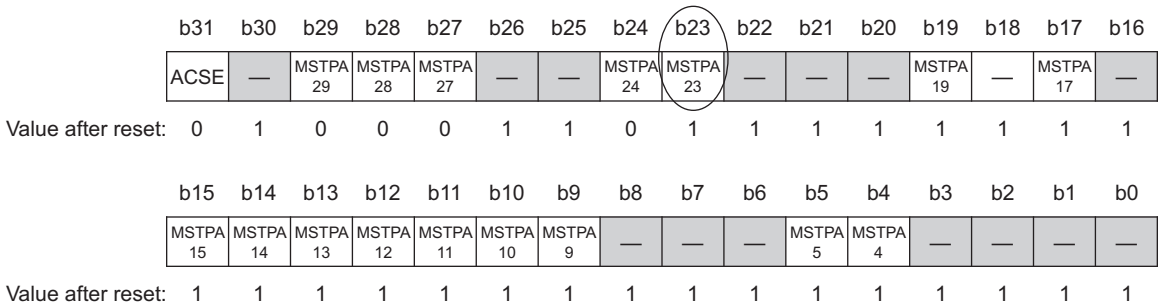
The width of the ADDR_n (16-bit) is greater than the width of the ADC output (10-bit). To avoid reading wrong data, the output has to be aligned either to the right or left of ADDR_n. This can be done by setting the ADCR2.DPSEL bit. This will be explained a little later.

4.3.2 Initializing the 10-bit A/D Converter

Module Stop Control Register A (MSTPCRA)

The module-stop control registers is a 32-bit register and can be used to place modules in and release modules from the module-stopped state. The several modules that realize frequency measurement are all stopped in their initial state. Releasing the modules from the stopped state makes operations for frequency measurement possible. Before we can utilize the 10-bit A/D converter, we would need to release the module from the stopped state by configuring bit 23 of the MSTPCRA[31:0] register.

Address(es): 0008 0010h



| | | | | |
|-----|---------|----------------------------------|--|-----|
| b23 | MSTPA23 | 10-bit A/D Converter Module Stop | Target module: AD | R/W |
| | | | 0: The module-stop state is canceled | |
| | | | 1: Transition to the module-stop state is made | |

Figure 4.22 Module Stop Control Register A (MSTPCRA) Description [1], page 282.

A/D Control/Status Register (ADCSR)

The Control/Status Register is used to select the input channels, start or stop A/D conversion, and enable or disable the ADI interrupt. The CH[2:0] register is used to select the analog channels which have to be A/D converted. The channels can be selected using the Table 4.6.

TABLE 4.6 Channel Selection [1], page 1690.

| WHEN ADCR.MODE[1:0] = 00B | | | | WHEN ADCR.MODE[1:0] = 10B OR 11B | | | |
|---------------------------|----|----|-----|----------------------------------|----|----|------------|
| B2 | B1 | B0 | | B2 | B1 | B0 | |
| 0 | 0 | 0 | AN0 | 0 | 0 | 0 | AN0 |
| 0 | 0 | 1 | AN1 | 0 | 0 | 1 | AN0, AN1 |
| 0 | 1 | 0 | AN2 | 0 | 1 | 0 | AN0 to AN2 |
| 0 | 1 | 1 | AN3 | 0 | 1 | 1 | AN0 to AN3 |
| 1 | 0 | 0 | AN4 | 1 | 0 | 0 | AN0 to AN4 |
| 1 | 0 | 1 | AN5 | 1 | 0 | 1 | AN0 to AN5 |
| 1 | 1 | 0 | AN6 | 1 | 1 | 0 | AN0 to AN6 |
| 1 | 1 | 1 | AN7 | 1 | 1 | 1 | AN0 to AN7 |

When used as an analog sensor, the QTI can detect shades of gray on paper and distances over a short range if the light in the room remains constant. The QTI sensor has 2 inputs and one output. When W is connected to V_{dd} (5V) and B is connected to V_{ss} (GND), the R terminal's voltage will drop or rise based on the shade of the surface. If all you want to know is whether a line is black or white, the QTI can be converted to a digital sensor by adding a 10 kΩ resistor across its W and R terminals. After doing so, the QTI behaves similarly to the circuit in figure 4.28. When W is connected to V_{dd} and B is connected to V_{ss} , the R terminal's voltage will drop below 1.4 V when the IR transistor sees infrared reflected from the IR LED. When the IR LED's signal is mostly absorbed by a black surface, the voltage at R goes above 1.4 V [7].

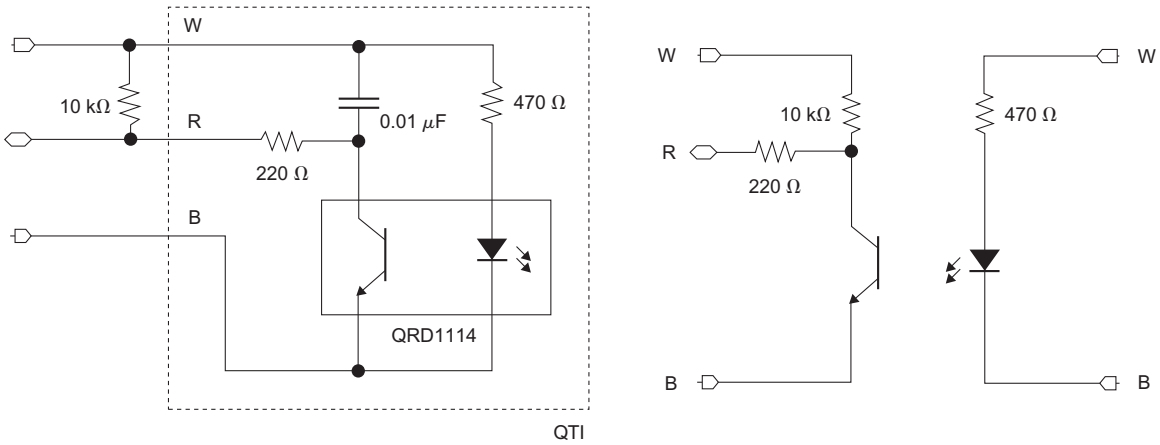


Figure 4.28 QTI sensor electrical characteristics using a 10K resistor [7].

Since we have made the design decision to use the QTI sensor, we have implied several derived requirements. (1) The surface of the environment will need to be white and (2) we will need to outline our borders in black. Can you think of any other requirements that can be derived from the design decision of using the QTI sensor to meet our initial set of requirements? Note that the QTI sensor needs to be very close to the surface/ground to get an appropriate reading.

Given the specification of the QTI sensor we know that our threshold voltage that will determine if the robot has reached the border will be 1.4V. We also know that our board V_{ref+} is 3.3V and V_{ref-} GND or 0V. Given this we can calculate the integer value for our threshold.

$$\left[\frac{1.4V(2^{10} - 1)}{3.3V} + \frac{1}{2} \right] \text{int} = 434 = 0x1B2$$

Index

A

A/D control register (ADCR),
129, 130
A/D control register 2 (ADCR2),
129, 131
A/D control/status register
(ADCSR), 127–28
A/D converter. *see* analog to
digital converter
A/D data registers (ADDRn),
125–26
ADC. *see* analog to digital
converter
ADCR, 129, 130
ADCR2, 129, 131
ADCSR, 127–28
ADDRn, 125–26
Algorithms
code, converting to, 31–33
code development,
description, 30
robotics application, 72
Aliasing, 119
ALU, 14
Analog to digital converter
10-bit converter (*see* 10-bit
A/D converter)
conversion rate, 118
description of, 117–19
digital value, of analog
value, 119
Nyquist frequency, 118–19
output of, 126
resolution of, 118
robotics application, 135–47
of RX63N microprocessor, 43
Sakura, port map for, 120–21
sample and hold circuit, 119
sensors, 135–47
voltage, reference, 118

Android system, 44
Applet, 44
Architecture, of code
development, 30
Arduino, 35, 36, 50
Arithmetic and Logic Unit
(ALU), 14

B

Baud rate clock, 78
BCLK, 40
Big up-front design, 27
Binary digit, 16
Bits, 16
Bitwise operations, 21
Bytes, 16

C

CAN, 3
Cascading timers, 87, 88–92
Central processing unit (CPU),
14–15, 38
Clock generation circuit, 40
Clock oscillator, 40
Clocks. *see also* timers
CMIEA Bit, 101, 112
CMIEB Bit, 101, 112
CMT, 41
Code
algorithm, converting to code,
31–33
algorithm, starting with, 30
coding standards, 32
description of, 30
design margins, tracking
of, 32
principles when
implementing, 32
robotics application, 72–74
Coding standards, 32

Communication function, of
RX63N, 42–43
Communications and
networking, 5
Compare match interrupt enable
A (CMIEA bit), 101, 112
Compare match interrupt enable
B (CMIEB bit), 101, 112
Compare match timer
(CMT), 41
Compilation mechanism, 22–23
Compiler
for embedded systems,
23–24
of GR Sakura, 44
Component costs, 3
Computer engineer, role of,
9–10
Computers, architecture of,
13–15
Const variable, 32
Constraints, of system
requirements, 29
Contact bounce, 60–61
Continuous scan mode, 10-bit
converter, 122
Control Area Network
(CAN), 3
Control mechanism, of
computer, 14
Control systems, 4
Control unit, 14
Controller Area Network (CAN)
module, 43
Conversion rate, of ADC, 118
Cost, of embedded systems,
1–2, 3, 6
CPU, 14–15, 38
Crank and start, 4
Customer requirements, 28