Using Interrupts With Peripherals

Chapter 9

Renesas Electronics America Inc.
Embedded Systems using the RX63N

Rev. 1.0

Interfacing with the Real World

- All embedded applications need to monitor some external events and process those events like a switch press for example.

- So how do we monitor an external event ?

Method 1. Polling

- We can *constantly monitor* all switches, and keep checking whether they are being pressed or not.
- Hence MCU will keep on polling the switches and wait unless it detects an event.
- Is this efficient?

Disadvantages of Polling

- The Answer is No.
- This is not efficient because it's a waste of CPU's resources (like power) to keep on waiting for a peripheral to respond.
- The code required for polling generally tends to be messy with multiple calls inside a big loop to check if a peripheral is ready or not.

Method 2: Interrupts

- The other method is : Peripheral sends a signal to CPU indicating an external event whenever it occurs.

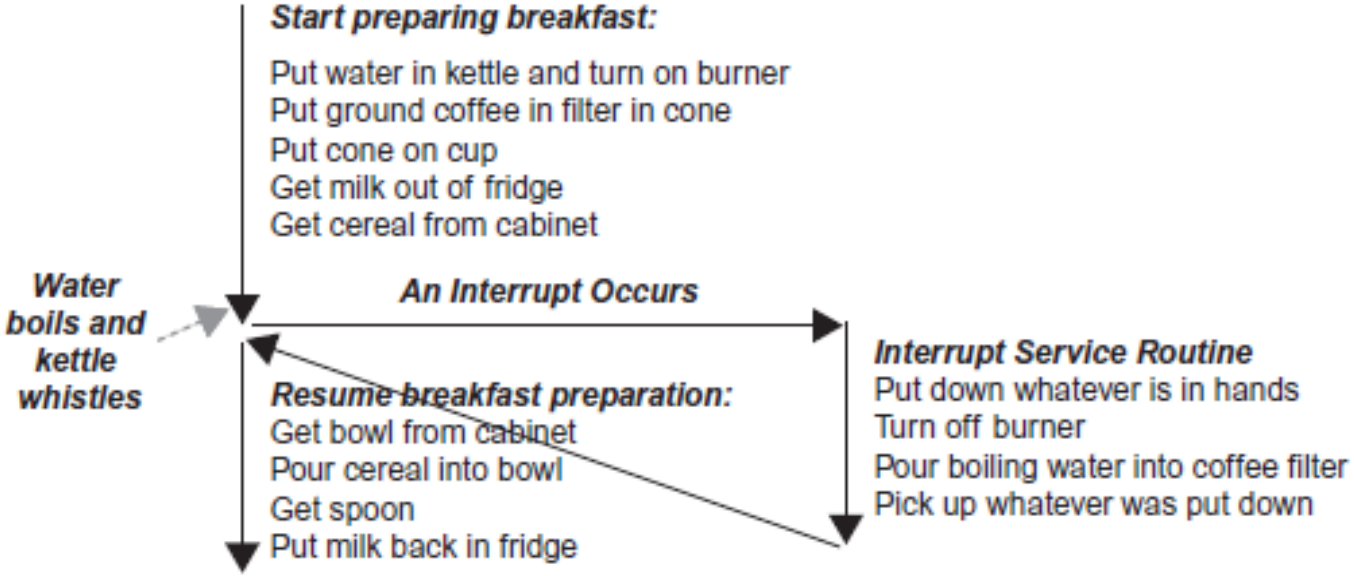
- CPU performs other tasks, and processes the external event *only* when it receives the signal from the peripheral.

Method 2: Interrupts

- The signal peripheral uses to draw the CPU's attention is called an ***Interrupt.***

- When CPU receives an interrupt, it stops its current execution, and processes the interrupt, and then resumes the original execution. This is called ***Interrupt Handling.***

Example



How CPU Processes the Interrupts

- **1.** Finish processing current instruction.
- **2.** Save program counter and flags to stack.
- **3.** Run interrupt service routine.
- **4.** Restore program counter and flags from stack.
- **5.** Resume main program.

Interrupt Service Routines (ISR)

- Subroutines used to service an interrupt are called **ISR.**
- Each interrupt has an ISR which has an address listed in ***Interrupt Vector Table***.
- Processor obtains the subroutine address from the vector table and directs the execution to the ISR.

Interrupt Vector Table

- Interrupt Vector Table (IVT) in the RX63N microcontroller has 256 interrupts.
- Each interrupt source occupies four bytes in the table.
- In total, the size of IVT is 1024 bytes (4 bytes, 256 sources).
- When the CPU accepts an interrupt, it acquires the 4-byte address from the IVT and executes the code specified at that particular interrupt service routine.

Types of Interrupts

- Interrupts from peripherals modules
- External pin interrupts (IRQ0 to IRQ15)
- Software interrupts
- Non Maskable interrupts

Non Maskable Interrupts

- NMI are interrupts that cannot be ignored by the processor.
- The non maskable interrupts are usually used for non-recoverable errors and serious hardware errors that need immediate attention.
- One example of a non-maskable interrupt is the reset pin.

Interrupt Request Detection Methods

- Edge detection

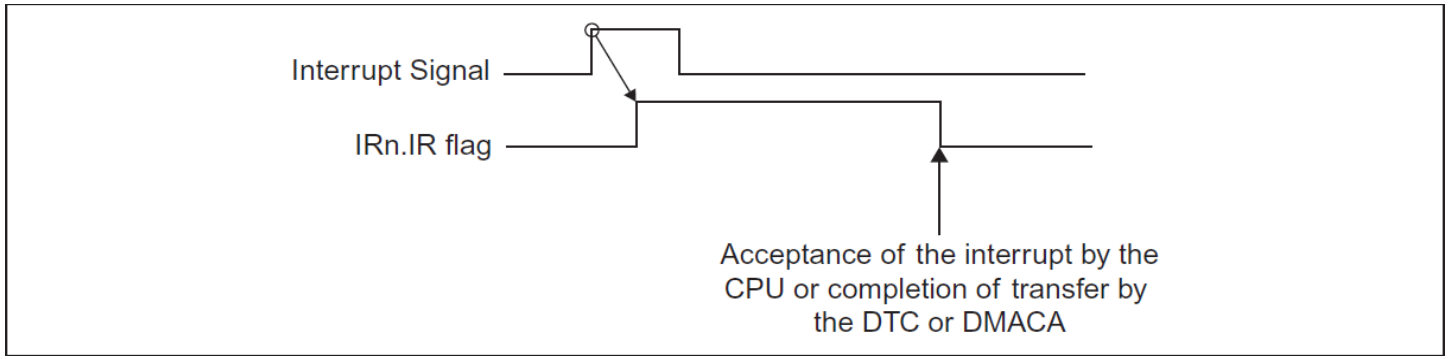
  The IR Flag is 1 when the interrupt is detected and is cleared by writing 0 to it once the request is accepted.

- Level detected

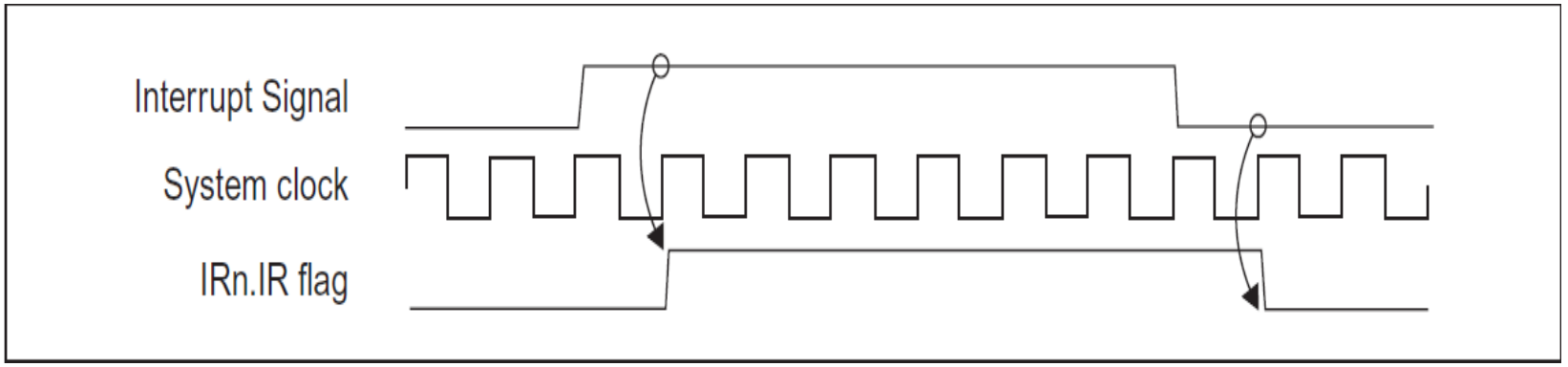
  The IR Flag is 1 when the interrupt is detected, but is cleared only when the interrupt source is cleared.

Interrupt Request Detection Methods

- Edge detection:



- Level Detection:



Registers

- Interrupt Request Register (IR)
- Interrupt Request Enable Register (IER)
- Interrupt Priority Register (IPR)
- Fast Interrupt Register (FIR)
- Software Interrupt Activation Register (SWINTR)
- IRQ Control Register (IRQCR)
- Non-Maskable Interrupt Status Register (NMISR)
- Non-Maskable Interrupt Enable Register (NMIER)
- Non-Maskable Interrupt Clear Register (NMICLR)
- NMI Pin Interrupt Control Register (NMICR)

Interrupt Request Enable Register (IERm)

- *m* 02 to 1Fh

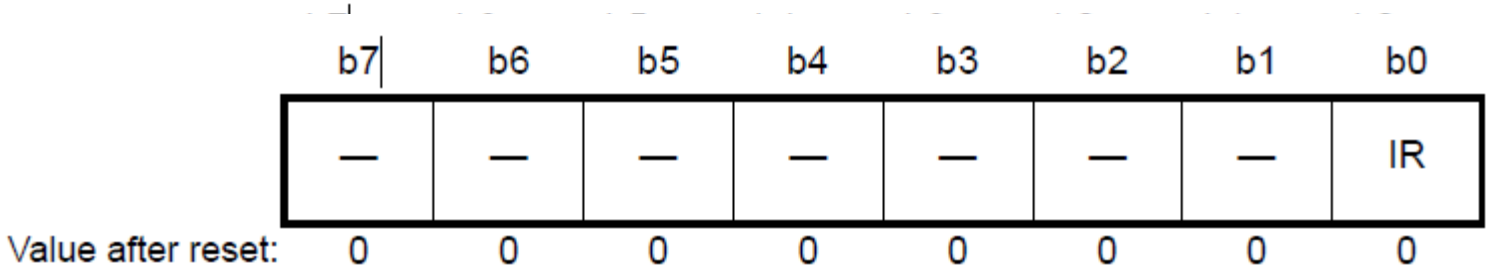
	b7	b6	b5	b4	b3	b2	b1	b0
	IEN7	IEN6	IEN5	IEN4	IEN3	IEN2	IEN1	IEN1
Value after reset:	0	0	0	0	0	0	0	0

- IREj (*j*=0 to 7) is used to enable or disable a particular interrupt. Writing 0 disables an interrupt, and writing 1 enables it.

- *m* and *j* values represent particular interrupts that can be referenced in the interrupt vector table.

Interrupt Request Register

- IRn where *n* represents the different interrupt sources in the interrupt vector table.
- Address(es): 0008 7010h to 0008 70FDh



	b7	b6	b5	b4	b3	b2	b1	b0
	—	—	—	—	—	—	—	IR
Value after reset:	0	0	0	0	0	0	0	0

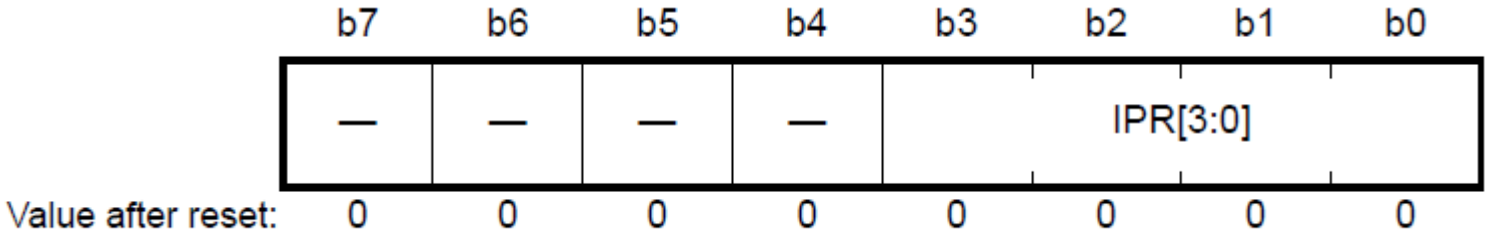
- Bit B0 represents the **interrupt status flag.**

  0: No interrupt request is generated.

  1: Interrupt Request is generated.

Interrupt Priority Register (IPRn)

- *n* goes from 0 to 253
- This register is used to set priority of an interrupt.

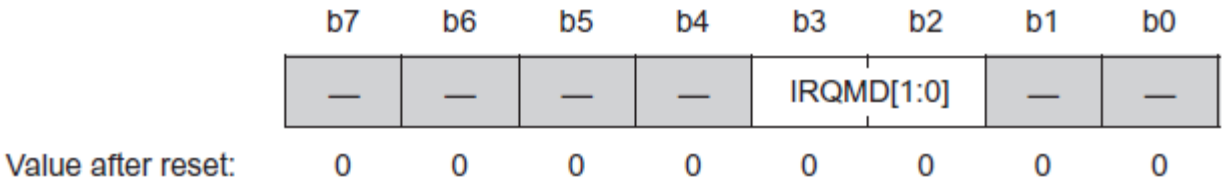


- Bit B3 to B0 are used to set the priority of the interrupt, 0000 being the lowest and 1111 being the highest.

- These bits can be written only when interrupt requests are disabled. IERm.IENj bit = 0

IRQ Control Register i (IRQCRi) (i=0 to 15)

- Address: 0008 7500h to 0008 750Fh



- The IRQ Control Register (IRQCRi) is used to select the type of external interrupts, which are either edge detection or level detection.

- These settings should only be done when the interrupt is disabled and after the interrupt is set up. The interrupt request bit for the particular external interrupt should be cleared and the interrupt enable bit should be set to 1.

Interrupt Controller Unit

- The part of the MCU that receives the interrupts from the peripherals is called the Interrupt Control Unit (ICU).

- ICU does the following:

1. Detecting the interrupts.
2. Enabling/Disabling interrupts.
3. Determining the interrupt destination.
4. Determining the priority.

Setting an Interrupt

The steps to set up an interrupt for a peripheral are as follows:

- The peripheral or port pin must be enabled and configured.
- Set an interrupt priority for the interrupt source (IPR macro) to a value greater than 0 (0 disabled).
- Enable the interrupt in the peripheral (local enable bit).
- Enable the interrupt in the ICU (IEN macro).

Setting an Interrupt

- Before enabling registers, the ISR for interrupt needs to be defined.
- Renesas provides the following macro for defining the ISR :

```
#pragma interrupt (SUBROUTINE_NAME(vect = VECTORNUMBER))

void SUBROUTINE_NAME(void);
```

This code is to be placed in the beginning of the program or in a separate ".h" file.

```
1. //Name of Interrupt
2. void SUBROUTINE_NAME(void){
3. //ISR Code
4. }
```

Setting an Interrupt

- All the registers can be accessed by macros defined by Renesas such as IPR(), IR().

- To clear the IR flag:

```
IR(“PERIPHERAL”, “INTERRUPT”) = 0;
```

- To check weather the flag is set or not:

```
if(IR(“PERIPHERAL”, “INTERRUPT”)) == 1)
{
//Code if flag is set
 }
```

Switch as Input using Interrupts

- The following code uses interrupts to signal a user input via a switch.

```
1. #include "iodefine.h"
2.
3. //Define SW1 interrupt
4. #pragma interrupt(SW1_Int(vect = VECT_IRQ8))
5. void SW1_Int(void);
6.
7. //Function definitions
8. void main(void);
9. void LED_Rotate(void);
10. void EnableIRQ8(void);
11.
12. //Define global variables
13. int Switch_Press = 0;
```

Switch as Input using Interrupts

```
14. int Current_LED
16. void main(void){
17.     ENABLE_LEDS
18.     EnableIRQ8();
19.     Current_LED = 1;
20.
21.         while(1){
22.             if(Switch_Press){
23.             LED_Rotate();
24.             Switch_Press = 0;
25.             }
26.         }
27. }
```

Switch as Input using Interrupts

```
1. void EnableIRQ8(void){
2.     IEN(ICU,IRQ8) = 0;
3.     PORT4.ICR.BIT.B0 = 1;
4.     ICU.IRQCR [8 ].BIT.IRQMD = 0x01;
5.     ENABLE_SWITCHES
6.     IPR(ICU,IRQ8) = 0x03;
7.     IR(ICU,IRQ8) = 0;
8.     IEN(ICU,IRQ8) = 1;
9. }
```

Switch as Input using Interrupts

- SW1 is connected to IRQ8, which is defined in the interrupt vector table at vector address 72.

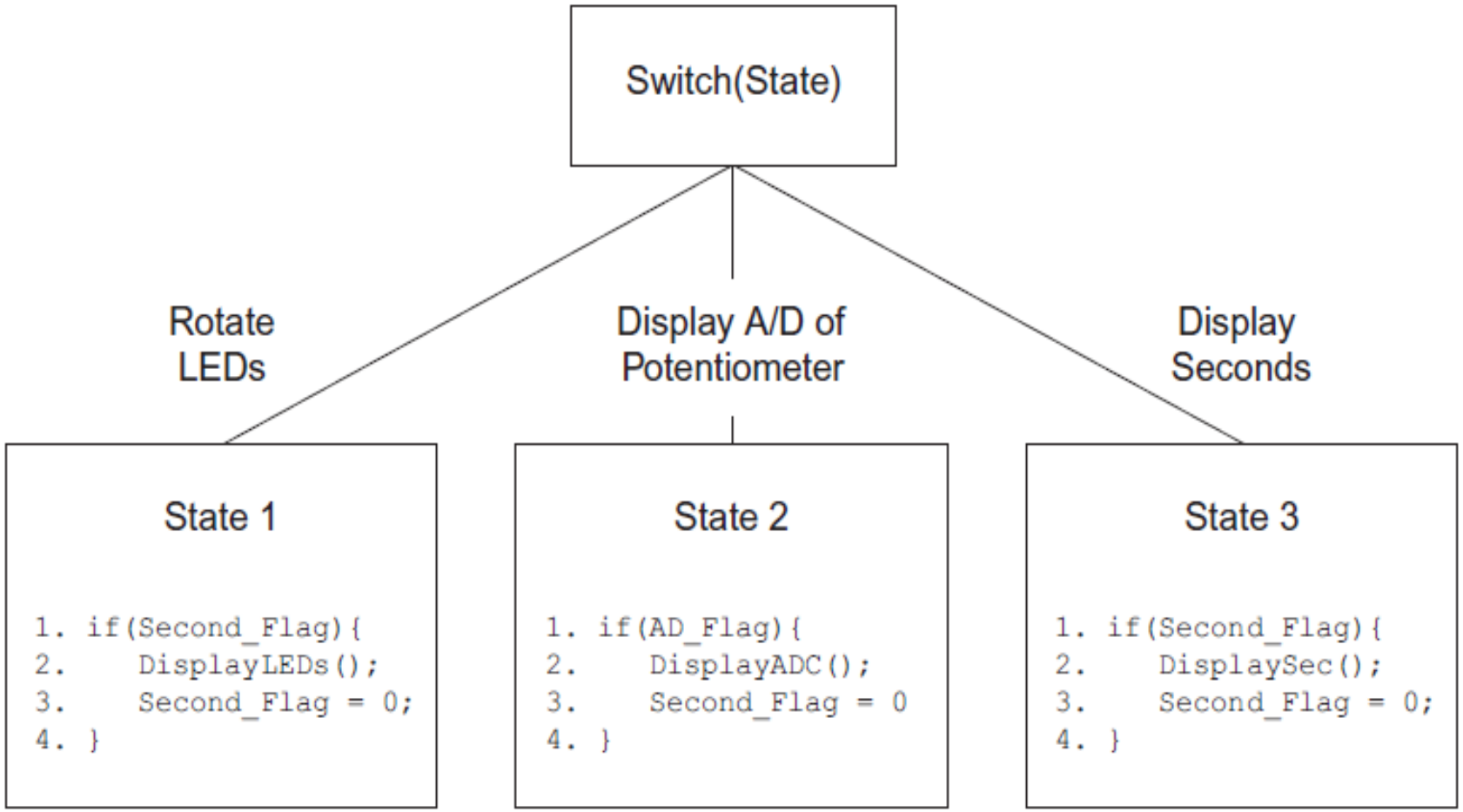
- The function enableIRQ8() interfaces the switch as an interrupt.

Using State Machines with Interrupts

- With state machines we think of our program as having ‘states.’
- We will implement the state machine by making a variable to represent the current state of the machine, and then using a switch statement to check that variable in the while(1) loop.

```
While(1)
{   …
       switch(state)
{    case s1:  //code
     case s2: //code
        .
        .
     case s3: // code }
```

Example of state machine with Interrupts



Example

- In one state the machine will rotate the LEDs in a circular sequence.
- In the second state we will display the value of the potentiometer on the LCD.
- In the third state the LCD will display the number of seconds elapsed since the program has started.
- Each time switch one is pressed, the board will cycle to the next state; once it goes beyond state three it will return to state one.

In This Chapter We Learned

- Polling
- Interrupts—How a CPU processes interrupts
- ISR
- IVT
- Edge and Level triggered interrupts
- Registers used for setting an interrupt
- Example: Switch as an input using interrupts
- Using state machines with interrupts

Renesas Electronics America Inc.