CMPE1250
Winter 2025
v1.1

ICA 7
Real-Time Interrupt

1 Introduction

Up to this point, you have only been able to create accurate timing by using `while()` loop blocking delays in your code. You should have seen in a previous ICA that these delays will be affected by changes in the bus speed. The length of your code's super loop will also change the timing of these blocking delays. Blocking delays will also prevent your code from doing anything productive while they are running. The responsiveness of whatever embedded system you are building will be severely impeded by long blocking delays.

In this assignment, you will be exploring the real-time interrupt (RTI) module. This will allow you to create extremely accurate timing intervals without the need to measure or calculate the execution time of a loop. While you will occasionally need to use blocking delays, the RTI will allow you to easily make non-blocking delays as well. The timing of these intervals will be dependant on the external oscillator on your microboard; changing the bus speed or the length of your super loop won't effect your delays. To do this, you will need to create a library to initialize the RTI. You will also need to create an interrupt service routine (ISR) to define what happens each time the interrupt is triggered.

2 Outcomes

This assignment is designed to assess your ability to:

- Build a reusable library to interact with the RTI
- Configure the RTI to create accurate time intervals
- Create an ISR that can function with the RTI
- Use non-blocking delays to accurately blink LEDs at different rates
- Discover how an ISR is executed
- Determine what the volatile and extern keywords do
- Infer the difference between a blocking delay and non-blocking delay

3 Assignment

In this assignment, you will be developing the *rti.c* file for your RTI library. This will involve writing one function to initialize the RTI and one function that will implement a blocking delay with the RTI. You will also create an ISR so you can do something useful when the RTI is triggered. You will be blinking LEDs in this assignment, but the RTI will be used to implement lots of features throughout the rest of your embedded courses.

3.1 Initialize the RTI

Create your *rti.c* file and define the `RTI_Init()` function. This function will need to initialize the interrupt to trigger every 1ms.

You may have noticed the following External Volatile Variables near the top of the *rti.h* file. These variables will need to be added to *rti.c* as global variables without the `extern` keyword. If you are unfamiliar with the `extern` or `volatile` keywords, please refer to Appendix A.

Listing 1: External Volatile Variables

```
//These variables have to be declared as global variables,
// but without the extern keyword in rti.c
extern volatile unsigned long rtiMasterCount;
extern volatile unsigned int rtiDelayCount;
```

Once your RTI initialization function has been written, you will need test it. This will require you to add an interrupt service routine to *main.c*. The ISR does not need to be prototyped because it should never be called by software. However, it does need a very specific declaration to tell the MC9S12 compiler that this function is the service routine for the real-time interrupt. An ISR Template with the correct declaration has been provided for you below.

All ISRs should begin by clearing the interrupt, or the program will get stuck in a loop forever running the ISR. The RTI service routine should also increment `rtiMasterCount` so you have a method of counting milliseconds as your program is running. For this assignment, also toggle the red LED in the ISR. This will give you a visual indicator that it is working. You can also measure the frequency of the LED to verify that the ISR is running every 1ms.

Listing 2: ISR Template

```
interrupt VectorNumber_Vrti void Vrti_ISR(void){
    //Clear the interrupt flag
    //Increment rtiMasterCount
    //Toggle the red LED
}
```

If your RTI is not working and you are confident that it has been initialized correctly, verify that you have uncommented the `EnableInterrupts;` line in *main.c*. If that line is commented out, all of the interrupts in your microcontroller will be disabled. This is a common mistake that is very easy to make. Make sure that you have also called the `RTI_Init()` function in the One-Time Initializations section of your code.

3.2 RTI Blocking Delay

Now that your ISR is working correctly, you can define the `RTI_Delay_ms()` function. This function will use the `rtiDelayCount` variable to implement a blocking delay. Depending on your implementation, you will need to increment or decrement the `rtiDelayCount` variable in your ISR. You will pass an integer into `RTI_Delay_ms()` and it will not return until that many milliseconds have passed. For example, `RTI_Delay_ms(50)` should create a 50ms blocking delay. Unlike your previous blocking delays, this function will not be effected by the bus speed or the length of your super loop.

Once your delay function is complete, use it to toggle the green LED every 50ms in the main loop. You should end up with a waveform as shown in Figure 1.

3.3 Long Delay with LED Select

Modify your main loop so the blocking delay is 1 second long. Also add functionality such that when the left button is pressed, the green LED turns off and the yellow LED toggles every 1 second. When the right button is pressed, the yellow LED should turn off and the green LED will begin toggling every 1 second again.

3.4 Button Responsiveness

In the previous section, you may have found that the buttons were very unresponsive. That is because they were only being checked once per second; they may need to held for a full second before the button press is recognized by the microcontroller. Modify your code to fix this issue. There are many ways to correct this problem, but you must end up with code that retains all the functionality of the previous section without blocking the microcontroller from checking the buttons. The buttons should seem very responsive to a user. Your solution should not cause the ISR execution time to greatly increase.

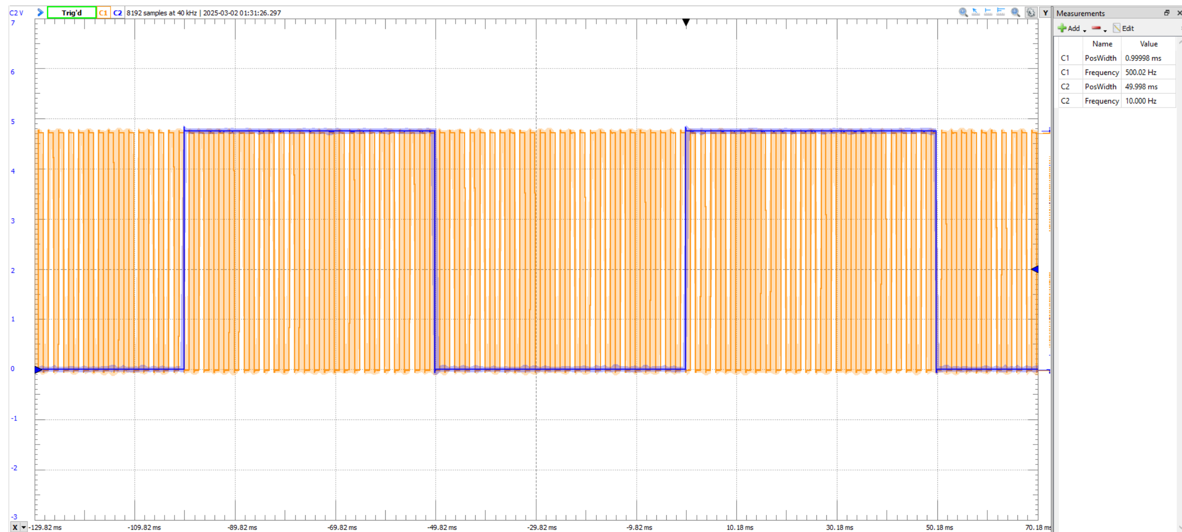

Figure 1: Part 2 Waveform

Appendices

A C Keywords

A.1 Extern

When you put `#include "rti.h"` in *main.c*, all of the contents of *rti.h* get pasted into your main C file at that spot. This is important to remember because it means that the code in the External Volatile Variables listing is actually a part of *main.c*; `rtiMasterCount` and `rtiDelayCount` are global variables in *main.c*. This is a good thing because you will need to access those variables from functions in your main C file. However, you will also need to access those variables from other C files.

The `extern` keyword tells the compiler that the variable already exists in another C file and that this declaration is just a reference to it. When you create your *rti.c* file, you will need to declare `rtiMasterCount` and `rtiDelayCount` without the `extern` keyword. This means that the actual variables will be declared and have memory allocated when *rti.c* is compiled. All of the other C files that need access to those variables have to declare them with the `extern` keyword. Because the variables are included in *rti.h* with the `extern` keyword, every compilation unit that includes *rti.h* will have access to those two variables.

A.2 Volatile

Interrupts allow functions to be called by hardware rather than being called in software. An interrupt could be triggered by a button press, a timer, a communication module, or any other number of things. This is great for us as embedded developers because it means that we don't need to poll registers to decide when to run some function. When the RTI triggers, your interrupt service routine will automatically run. You can also use interrupts to automatically handle data on the serial port. If you use the microcontroller's analog to digital converter, you can have an interrupt trigger when it is done converting a measurement. There are lots of uses of interrupts. They can make your code run faster and they can allow you to create low-power embedded devices. The microcontroller can sit in a low-power mode waiting for something to happen rather than constantly checking to see if it has happened.

Interrupts are a nightmare from the compiler's perspective. The programmer has given it functions

that are never called in software. There are variables that are read from, but are only written to in these functions that are never called. When the compiler tries to optimize your code, it won't know what to do with all of these variables that need to be accessed at random and outside of the normal program flow. It will try to save RAM by converting them to constant values, but then the developer (you) starts getting upset about it. The `volatile` keyword lets the compiler know that these variables are volatile; they're likely to change or be accessed at any time without notice, even if it's outside of the normal program flow. The `volatile` keyword is your way of telling the compiler that it can't optimize some variables. The memory for those variables needs to be permanently reserved in RAM and it always needs to be accessible. If you use a variable in an ISR, it should be declared with the `volatile` keyword.

A.3 Further Readings

GeeksForGeeks: Understanding the `extern` Keyword in C
GeeksForGeeks: How to Use the `volatile` Keyword in C

B RTI Header File

```c
//RTI Module Library
//File: rti.h (header file)
//Processor: MC9S12XDP512
//Crystal: 16 MHz
//by Carlos Estay
//September 2023
//Last edit, September 13th, 2023


//These variables have to be declared as global variables,
// but without the extern keyword in rti.c
extern volatile unsigned long rtiMasterCount;
extern volatile unsigned int rtiDelayCount;


/// @brief Enables RTI Module
/// @param
void RTI_Init(void);

/// @brief Blocking delay to be used once the RTI Module is enabled
/// @param timeout
void RTI_Delay_ms(unsigned int timeout);



/*The following 2 functions will be implemented in 2250*/
/// @brief Enables RTI Module with callback to be used in main
/// @param function
void RTI_InitCallback(void(*function)(void));

/// @brief Enables RTI Module with callback to be used in main and called
/// every "x" milliseconds
/// @param function
/// @param interval in [ms]
void RTI_InitCallback_ms(void(*function)(void), unsigned int);
```