Interrupts Using the Multi-Rate Timer (MRT)

1 Introduction

Timers are one of the most important peripheral device types on a microcontroller, because they allow precise timing of events. The timer typicallt has a constant frequency clock input, a prescaler and a counter. A pre-calculated value is written into the prescaler which divides the input clock frequency by that value. The output of the prescaler, the lower frequency signal becomes the clock signal of the counter. As the counter counts up from zero, it is compared with another pre-calculated value. When a match occurs, an event flag is raised, the counter is loaded with zero and starts counting again.

The match event is calculated to precisely time periodic events; that means, when the flag of the timer is raised, the processor should run a pre-determined response function, such as read an analog voltage, or set a GPIO pin etc. Typically a match event can be used to raise an interrupt and thus the response function can be configured as an interrupt service routine (ISR).

All timers can be configured to generate a periodic interrupt using the mechanism above. Specifically, the MRT of LPC824 does not have a prescaler. However it is 24 bits wide, so that it can count up to about 167,000,000, so that a prescaler is typically not needed.

The SysTick timer mentioned elsewhere in this week's lecture notes is similar. The SysTick timer is a specification of ARM, and therefore it is implemented exacly the same in all ARM processors regardless of manufacturer. However, MRT is a peripheral device that was implemented by NXP. Other manufacturers have timers with similar functionality but not exactly the same as MRT.

2 ISR Generated by MRT

The MRT project is a simple demonstration program of the SDK MRT driver. It sets up the MRT hardware block to trigger a periodic interrupt two times per second (every 500ms). When the MRT interrupt is triggered, the LED is toggled on Alakart.

Open the file `mrt.c` and follow through with the explanations below and the comments in the code:

In `main()`:

- Initialize the CPU clock:

```
clock_init();  // Initialize CPU clock to 30MHz
```

- Initialize the LED pin as an output: Here, "kCLOCK_Gpio0" is a constant defined in the file `fsl_clock.h`.

```
// Initialize the GPIO pin where the LED is connected:
CLOCK_EnableClock(kCLOCK_Gpio0);
GPIO_PinInit(GPIO, LED_PORT, LED_PIN, &led_pin_conf);
```

- Enable the clock of MRT

```
CLOCK_EnableClock(kCLOCK_Mrt);  // Enable clock of mrt.
```

- Query and initialize the MRT to default values:

```
MRT_GetDefaultConfig(&mrtConfig);
MRT_Init(MRT0, &mrtConfig);
```

- Set MRT to periodic INT generation mode. `Sec. 19.5.1` has detailed information about this.

```
// Setup Channel 0 to periodic INT mode (See Sec. 19.5.1)
MRT_SetupChannelMode(MRT0, kMRT_Channel_0, kMRT_RepeatMode);  //
 Init MRT module to default configuration.
```

- Calculate the value to count up to. In our case, the processor bus main clock is 60MHz. We desire INTs at 2Hz, so we must count up to 30,000,000. We can do two things: We can write 30,000,000 as `mrt_count_val`. If we later decide to run the processor slowly, the timer will produce INTs at a different rate. The other thing we can do is, to check the current value of the MRT clock and the desired INT frequency, and directly calculate the value. We do the second in this program.
  To get `DESIRED_INT_FREQ` number of INTs per second, Channel 0 of MRT must count up to: `(Input clock frequency)/ (desired int frequency)`. (Input clock frequency in this case is 60,000,000. For 2 INT per second, the count value calculates as 30,000,000.) **CAUTION**: MRT counter is only 24 bits wide. See Sec. 19.6.2 Timer register and 19.6.5 Module Configuration register. However, this still means we can write up to about 167,000,000 here, and this is sufficient for most cases.

```
// Query the input clock frequency of MRT
mrt_clock = CLOCK_GetFreq(kCLOCK_CoreSysClk);
mrt_count_val= mrt_clock/DESIRED_INT_FREQ;
```

- We start the MRT timer Channel 0 with the calculated count value:

```
MRT_StartTimer(MRT0, kMRT_Channel_0, mrt_count_val);
```

- Finally we enable MRT interrupts for channel 0. See Sec. 19.6.3 Control register.

```
MRT_EnableInterrupts(MRT0, kMRT_Channel_0,
 kMRT_TimerInterruptEnable);
```

- Enable the interrupts in the NVIC. This allows the microcontroller to respond to MRT INT events. See Sec. 4.4.1 Interrupt Set Enable Register 0 register, Bit 10.

```
EnableIRQ(MRT0_IRQn);  // Enable MRT INTs in NVIC.
```

NOTE: As always, constants or structures starting with a 'k' such as `kMRT_Channel_0` are defined in the device driver libraries of Xpresso SDK.

3 The Vector Table

There is one more important thing to do: **We must link the interrupt signal with the interrupt service routine (ISR)**. To do that, we must write the pointer to the ISR function at the correct location of the interupt vector table. MRT INT vector is the 10th element of the vector table (26th entry from the begining).

The function pointer is simply the name of the function. The ISR vector is set inside `startup_LPC824.S` file as:

```
.long   MRT0_IRQHandler                 /* Multi-rate timer interrupt */
```

Finally, the ISR function itself must be written. This is again in the file `mrt.c` as:
`"void MRT0_IRQHandler(void)"`
Obviously, it must have the same name as the pointer in the vector table.

When the MRT INT occurs, it sets the global variable `mrtIsrFlag=true`. Within `main()`, we keep checking `mrtIsrFlag` and when it is true we toggle a LED.

The ISR looks like this:

- An ISR does not have any calling arguments nor return values. It is called by the hardware, and it does not return to any specific location in the code. Therefore the argument list must be `void` and the return type must also be `void`.

```
void MRT0_IRQHandler(void) {
```

- It first clears the INT flag of MRT channel 0:

```
MRT_ClearStatusFlags(MRT0, kMRT_Channel_0,
 kMRT_TimerInterruptFlag);
```

- Then it sets the global variable to "`true`"

```
1   mrtIsrFlag = true;
```

This is a detailed description of the program and how interrupts are serviced. When the code is compiled and ran, the Blue LED on Alakart will start to flash at a period of 1s.

A.O. 2023