Emote Internals

|  |  |  |
| --- | --- | --- |
| Version | Author | Description |
| 0.0 | Nived.Sivadas | Seminal Version |
|  |  |  |

Chapter 1

**Introduction**

This chapter introduces the eMote. The purpose of this chapter is to provide a brief overview of the different drivers used by the eMote and their location in the source tree. This document only provides information on the actual porting and ignores details about the hardware itself which can be found in the reference manual.

1.1 Driver List

The eMote represents samraksh’s port of the .NET Micro framework also called NETMF. The eMote in addition to containing Samraksh’s port of the STM32 also contains modifications like real time extensions. The eMote runs on a device called the .NOW at the time of writing of this document. The eMote through various drivers provides C# users with the ability to interact with both on board peripherals on the .NOW like the RF231 radio, lcd and external peripherals attached to the .NOW through the sensor board and long range radio expansion boards.

1. Interrupt Controller
2. Timers
3. Advanced Timer
4. Time
5. USART
6. Radio
7. Long Range Radio
8. LCD
9. External NOR
10. SD Card
11. ADC Sampling burst mode extensions
12. DAC driver
13. Accelerometer
14. Temperature Sensor

Chapter 2

**Interrupt Controller**

Location:

$(SPOCLIENT)\DeviceCode\Targets\Native\STM32F10x\DeviceCode\drivers\intc

Where $(SPOCLIENT) represents the root path of NETMF.

Source Files:

*STM32.h*

Contains the interrupt controller class and register description.

*STM32\_\_AITC.cpp*

Contains the implementation details for functions like activate, enable and disable interrupts for different interrupts supported under the Cortex M3.

*STM32\_nvic\_function.cpp*

Contains the glue code that connects the interrupt controller driver with the expectations of NETMF.

*CPU\_INTC\_Initialize*

This function is responsible for initializing the interrupt controller and is called from BootEntry in file tinyhal.cpp. Internally calls STM32\_AITC\_Driver::Initialize. In addition to clearing all enabled interrupts and pending interrupts, this function also sets the priority of each of the interrupts.

Prioirty

The Cortex M3 supports two types of priorities called pre - emption priority and sub – priority. Whoever has higher pre – emption priority is executed first. If there are 2 interrupts with the same pre – emption priority, then the one who has the higher sub priority will be executed first. If both interrupts have the same pre-emption and sub priority, the one that comes first is executed first.

The eMote configures the INTC with NVIC\_PriorityGroup\_4 which means that all 4bits of the IP register are configured for pre – emption priority and there are 0 bits for sub – priority.

The priorities are statically defined in the vector table STM32\_AITC\_Driver::s\_IsrTable which also holds pointers to callbacks and can also be changed dynamically using the SetPrority interface. NETMF does not support changing the priorities dynamically.

CPU\_INTC\_ActivateInterrupt

Responsible for activating an interrupt and assigning a callback handler when the interrupt is generated. ActivateInterrupt also automatically enables the interrupt.

Interrupt Vector Table

The table is found in file STM32\_\_AITC.cpp. It contains the interrupt vector and the priority assigned to it at compile time. The table also contains a pointer to the function handler that gets assigned to it during a call to ActivateInterrupt.

The rest of the functions of the interrupt controller involve straight forward register manipulation and should be easy to understand.

Chapter 3

Timer and Time drivers

The STM32F10x processor contains about 4 general purpose timers, 2 advanced timers and a few other special timers. The older timer drivers were based on the general purpose timers which were all 16bit timers. This meant that at a clock speed of 48MHz, a rollover happened on 1.3ms which primarily meant that the .NOW could not sleep for more than that period of time.

This system was then replaced by the advanced timer system which consists of a 16 bit advanced timer chained to a 16 bit general purpose timer. This simulates a 32 bit timer giving the ability to sleep much longer.

Location (old timers):

$(SPOCLIENT)\DeviceCode\Targets\Native\STM32F10x\DeviceCode\drivers\tim

Location (advanced timer):

$(SPOCLIENT)\DeviceCode\Targets\Native\STM32F10x\DeviceCode\drivers\advancedtim

Source:

*netmf\_advancedtimer.cpp*

*netmf\_advancedtimer.h*

*netmf\_time\_functions.cpp*

The netmf\_time\_functions.cpp contains the glue code needed to link the time api expected by the NETMF with the advanced timer system.

TIM1 (advanced timer) and TIM2 (general purpose timer) are the two timers involved in the chaining. Every time the advanced timer overflows it sends an update event in hardware to the general purpose timer which then increments its counter. The timers together therefore achieve a 32 bit timer.

SetCompare:

The other advantage of having a 32 bit timer is that you can set 32 bit compare values. The most significant 16 bits of the compare value are set in TIM2 when this fires, the handler now sets the least significant 16 bits in TIM1.

The users of the advanced timer are notified of the compare event through a software interrupt (tasklet). This isolates the user notification from the timer handler, thereby reducing jitter of the timer system.

Chapter 4

Virtual Time Layer