www.qorvo.com
lpw.support@qorvo.com

gpSched Reference Manual

API Description

Version latest
December 13, 2017

Contents

Chapter 1

Introduction

The *gpSched* component is a light-weight OS implementation used within the Qorvo stack. It works as a cooperative scheduler, executing function callbacks at set timings.

1.1 Design

1.1.1 Main loop

The main() function is included in this component. The main body will:

1. Execute polled actions: Radio bottom-half interrupt handling, serial command interpretation
2. Look up events in it's event list.
3. Call the callback function of the first event that reached his execution time.
4. Check if events are pending within a GoToSleep threshold. If events are further of, the system will be set to sleep until the nearest event.

1.1.2 Interrupt handling

The scheduler is intended to use a top-half/bottom-half handler approach for interrupt handling. Care should be taken to limit the time spent in interrupts. Further handling of interrupts should be pushed to scheduled events. This top-half/bottom-half handler approach limits the time spent in atomic sections and keeps an embedded system reactive to the higher priority items.

```
void InterruptA(void)
{
    //Disable interrupt mask to avoid re-trigger
    InterruptA_Disable();

    //Schedule handling
    gpSched_ScheduleEvent(0, InterruptA_Handler);
}

void InterruptA_Handler(void)
{
    //Do slow handling
    ...

    //Re-enable interrupt mask
    InterruptA_Enable();
}
```

1.2 Intended use

1.2.1 Waiting

Functions ran by the scheduler should never block as they will block the entire execution queue. If a wait is required, a timeout function should be scheduled to take the appropriate actions. The timeout function can be unscheduled when the criteria is met or used as a flag for other actions by checking its existence.

Chapter 2

File Documentation

2.1 gpSched.h File Reference

The PeakNet scheduler handles the time of a PeakNode.

Macros

- #define GP_SCHED_TIME_RES_ORDER 5
- #define **GP_SCHED_TIME_RES** (1<<GP_SCHED_TIME_RES_ORDER)
- #define **GP_SCHED_US2TIME**(us) (us >> GP_SCHED_TIME_RES_ORDER)
- #define **GP_SCHED_DISTANTEVENTS_INTERVAL** (20∗60)
- #define **GP_SCHED_EVENT_TIME_NOW** ((UInt32) -1L)
- #define **GP_SCHED_EVENT_TIME_MAX** 0x7FFFFFFF
- #define GP_SCHED_TIME_COMPARE_LOWER_US(t1, t2) (!((UInt32)((t1) - (t2))/∗&(0xFFFF-FFFF)∗/ < (0x80000000LU)))

  Compares times from the chip's timebase - check if t1 < t2 (in us)
- #define GP_SCHED_NO_EVENTS_GOTOSLEEP_THRES ((UInt32)(0xFFFFFFFF))

  Allow sleep only if no events are pending.
- #define GP_SCHED_DEFAULT_GOTOSLEEP_THRES GP_SCHED_NO_EVENTS_GOTOS-LEEP_THRES

  Default time between events before going to sleep is considered.
- #define MAIN_FUNCTION_NAME main

  the MAIN_FUNCTION_NAME define will be used as entry point for the gpSched scheduler

Typedefs

- typedef void(∗ **gpSched_EventCallback_t**)(void ∗)
- typedef Bool(∗ **gpSched_GotoSleepCheckCallback_t**)(void)

Functions

- void gpSched_Init (void)

  Initializes the scheduler.
- void **gpSched_DeInit** (void)
- Bool gpSched_SetGotoSleepThreshold (UInt32 Threshold)

  Sets the minimum time required between events for going to sleep.

- void **gpSched_SetGotoSleepCheckCallback** (gpSched_GotoSleepCheckCallback_t gotoSleep-CheckCallback)
- void **gpSched_SetGotoSleepEnable** (Bool enable)
- void **gpSched_GoToSleep** (void)
- void gpSched_StartTimeBase (void)

  Starts the time base.
- UInt32 gpSched_GetTime (void)

  Returns the current time of the time base.
- Bool gpSched_TimeCompareLower (UInt32 n_time1, UInt32 n_time2)

  Compare 2 times (in 1us) taking overflows into account.
- void gpSched_Clear (void)
- Bool gpSched_EventQueueEmpty (void)

  Returns the event queue status.
- void gpSched_ScheduleEvent (UInt32 rel_time, void_func callback)

  Schedules a scheduled event.
- void **gpSched_ScheduleEventArg** (UInt32 rel_time, gpSched_EventCallback_t callback, void ∗arg)
- Bool gpSched_UnscheduleEvent (void_func callback)

  Unschedule a scheduled event.
- Bool gpSched_ExistsEvent (void_func callback)

  Check if an event is still waiting for execution.
- UInt32 gpSched_GetRemainingTime (void_func callback)

  Get the remaining time untill event execution.
- Bool gpSched_UnscheduleEventArg (gpSched_EventCallback_t callback, void ∗arg)

  Unschedule a scheduled event.
- Bool gpSched_ExistsEventArg (gpSched_EventCallback_t callback, void ∗arg)

  Check if an event is still waiting for execution.
- UInt32 gpSched_GetRemainingTimeArg (gpSched_EventCallback_t callback, void ∗arg)

  Get the remaining time untill event execution.
- void gpSched_DumpList (void)

  Debug function to dump Event content.
- void **gpSched_Main_Init** (void)
- void **gpSched_Main_Body** (void)

2.1.1 Detailed Description

The time base in this scheduler has the following structure:

The base period is split into a number of intervals (see function gpSched_SetPeriod()). Each interval is split into 2 subintervals.

The timing of the intervals and subintervals is defined by the asynchronious timer of the AtMega, i.-e. the asynchronious timer interrupt is used to indicate the beginning of the interval or subinterval. The hardware Timer1 (Atmega) is used to measure the time within one interval. This time measurement restart in the beginning of each interval (not reset on the second subinterval). The timer1 has a discontinuity at the end of each interval. The Timer1 is started in the ISR of the asynchronious timer.

A function call that should be triggered by the scheduler in the future is called 'event'. Two different types of events exists: slotted and unslotted events.

Unslotted event (see function gpSched_ScheduleEvent() and gpSched_ScheduleEventArg()) are executed from the main loop and have the lowest priority. Slotted events are triggered by an interrupt and have a higher priority (see gpSched_ScheduleSlottedEvent() and gpSched_ScheduleSlotted-EventArg()). The interrupt frequency is 320 us.

Drift prediction

The drift prediction is used to handle the clock drift of the asynchronious timer between different devices. The following assumptions are made:

The Timer1 runs on the same (absolute) frequency on all devices and the measurement made based on this timer are accurate. The asynchronious timer does not run at the same frequency on all devices.

The drift of the asynchronious timer is measured using a synchronisation signal. To compensate the drift, a correction is applied to the timer1 offset and to the asynchronious timer itself in the ISR of the asynchronious timer before starting timer1.

2.1.2 Macro Definition Documentation

#define GP_SCHED_TIME_RES_ORDER 5

System Time. The Scheduler works with 32us resolution.

#define MAIN_FUNCTION_NAME main

When defining MAIN_FUNCTION_NAME to a certain function, another main() implementation is expected To start the normal gpSched operation, the MAIN_FUNCTION_NAME can be called. gpSched will then loop infinitely, executing any scheduled event and servicing stack functionality

2.1.3 Function Documentation

void gpSched_Clear (void)

Clears the event queue.

void gpSched_DumpList (void)

GP_SCHED_DIVERSITY_DEBUG must be enabled.

Bool gpSched_EventQueueEmpty (void)

Returns the next event to execute. The return pointer is NULL when no event is pending yet. This function should be called in a polling loop in the main thread.

Returns

Event queue status:

- true: Queue is empty
- false: Still events in the queue

Bool gpSched_ExistsEvent (void_func *callback*)

Parameters

callback	Scheduled event callback to check.

Returns

True if callback is still pending.

Bool gpSched_ExistsEventArg (gpSched_EventCallback_t *callback,* void ∗ *arg*)

Parameters

callback	Scheduled event callback to check.
arg	Pointer to the the arguments used when scheduling. Use NULL to skip matching the arg pointer.

Returns

True if callback is still pending.

UInt32 gpSched_GetRemainingTime (void_func *callback*)

Parameters

callback	Scheduled event callback to check.

Returns

Time in us untill event normally executes. If event is not found 0xFFFFFFFF is returned.

UInt32 gpSched_GetRemainingTimeArg (gpSched_EventCallback_t *callback,* void ∗ *arg*)

Parameters

callback	Scheduled event callback to check.
arg	Pointer to the the arguments used when scheduling. Use NULL to skip matching the arg pointer.

Returns

Time in us untill event normally executes. If event is not found 0xFFFFFFFF is returned.

UInt32 gpSched_GetTime (void)

Returns

The current time in time base units of 1us.

void gpSched_Init (void)

This function intializes the scheduler.

void gpSched_ScheduleEvent (UInt32 *rel_time,* void_func *callback*)

Schedules an event. The event is inserted into the event queue.
Parameters

rel_time	Relative execution time (delay) in us. If the delay equals 0, the function will be scheduled for immedate execution. rel_time shall not exceed GP_SCHED_EVENT_TIME_MAX.

callback	Callback function.
arg	Pointer to the argument buffer.

Bool gpSched_SetGotoSleepThreshold (UInt32 *Threshold*)

This function sets a threshold in us, that is used for determining if we can go to sleep. If the next scheduled event is at a time later than the current time + the defined threshold (- wakeup threshold) then going to sleep is allowed
Parameters

Threshold	threshold in us

void gpSched_StartTimeBase (void)

Starts the time base, i.e. the timer interrupts are enabled.

Bool gpSched_TimeCompareLower (UInt32 *n_time1,* UInt32 *n_time2*)

Parameters

n_time1	Time (in us) to be compared.
n_time2	Second time (in us) to compare against n_time1.

Returns

return True if n_time1 is earlier then n_time2

Bool gpSched_UnscheduleEvent (void_func *callback*)

Unschedules a scheduled event before its execution. The event is identified with the callback pointer
Parameters

callback	Scheduled event callback.

Returns

True if event was found. False if event was no longer in the queue.

Bool gpSched_UnscheduleEventArg (gpSched_EventCallback_t *callback,* void $*$ *arg*)

Unschedules a scheduled event before its execution. The event is identified with the callback pointer and the arguments content.
Parameters

callback	Scheduled event callback.
arg	Pointer to the the arguments used when scheduling. Use NULL to skip matching the arg pointer.

Returns

True if event was found. False if event was no longer in the queue.