HostService Objects

Introduction

A **HostService** object allows users to synchronize their application to the "heartbeat" of the controller. It can be useful to monitor the health of the DSP, as a deterministic non-busy application delay, or even to modify SynqNet packets in real-time.

XMP and ZMP-SynqNet series motion controllers have the ability to interrupt the host at a specified frequency, where the frequency is a multiple of the controller's sample rate. This board-to-host interrupt is referred to as the Sync Interrupt. This object spawns a thread, which waits for the Sync Interrupt and then executes a user defined *Sync Interrupt Service Routine* (or SISR) in the user space.

The primary use for the HostService object is for host systems that need to process incoming SynqNet data from the controller and then write data back to the controller before a SynqNet transmission occurs. This is a convenient way to calculate control loops or inject values into SynqNet data packets in real-time. This type of operation will usually require a real-time operating system (VxWorks and RTX/RTSS).

The controller also has an optional watchdog timer, named the HostProcessFlag, which verifies that data written by the SISR is valid before it's sent over the SynqNet network.

The hostService object is provided as an interface to the Sync Interrupt feature. It provides code for thread-creation/deletion, setting/clearing the HostProcessFlag, and other common operations required for proper Synq Interrupt processing.

For more information about Sync Interrupt, please see the Synq Interrupt page.

NOTE: The HostService object is not part of the standard MPI. In order to use the Service Object, the *apputil.h* file needs to be included by your code and the apputil library needs to be linked to your application. In addition, thread handling is something that is different on every operating system. Therefore, HostService objects may have different behaviors on different operating systems. Programmers that are experienced in multi-threaded application programming may want to program their own host service SISR threads (see sample application syncInterrupt.c for a Win32 example). The source-code for this object is available in the (c:\MEI)\apputil project directory.

Methods

Create, Delete, Validate Methods

hostService**Create**	Create a HostService object, an SISR thread, and attach a serviceFunction for a given Control object.

hostService**Delete**	Stop SISR thread and delete the HostService object.

Configuration and Information Methods

hostService**Enable**	Enable or disable the HostService SIST thread from executing the serviceFunction. Disabled by default.
hostService**Status**	Returns status structure for HostService object.

Data Types

HostService**Function**

HostService**Status**

hostServiceCreate

Declaration

```
const HostService hostServiceCreate(MPIControl             control,
                                    HostServiceFunction    serviceFunction,
                                    void                   *params,
                                    long                   interruptPeriod,
                                    MPI_BOOL               enableWatchdog,
                                    long                   priority)
```

Required Header: service.h

Description

hostServiceCreate creates an SISR thread running at the specified ***priority*** for the given ***control*** object that executes the ***serviceFunction*** every ***interruptPeriod*** samples. After the create is called, the Sync interrupt occurs. By default, the service function is disabled from executing. Use hostServiceEnable(...) to enable the execution of the ***serviceFunction***.

control	a handle to a Control object.
serviceFunction	a handle to a user defined Sync Interrupt Service Routine (SISR) function.
***params**	points to a user defined structure that will be passed to **serviceFunction**.
interruptPeriod	specifies the frequency of the Sync Interrupt (where the frequency is a multiple of the controller’s sample rate). This value must be greater that zero. A value of **1** generates an interrupt every controller sample. A value of **2** generates an interrupt every other sample, etc.
enableWatchdog	if TRUE, the SISR thread sets the HostProcessFlag watchdog flag in the firmware just before executing the **serviceFunction**. When the **serviceFunction** is complete, the SISR thread will clear the watchdog flag. If the watchdog flag is still set when the firmware starts SynqNet transmission, the controller will generate a MEIEventTypeCONTROL_HOST_PROCESS_TIME_EXCEEDED event. The event indicates that the host's serviceFunction did not complete before SynqNet transmission. This should be used if the host is trying to (for example) process feedback and calculate a new DAC command for a drive located on the network.

priority	is a platform specific variable. *If "priority" is* → *Then*: -1 → hostServiceCreate will attempt to assign the maximum priority to the hostService thread. >0 → hostServiceCreate will attempt to assign the priority to the hostservice thread. **NOTE**: To avoid CPU usage competition, it is recommended that you run this hostService thread at a priority slightly higher than the apputil service thread.

If "priority" is	***Then***
-1	hostServiceCreate will attempt to assign the maximum priority to the hostService thread.
>0	hostServiceCreate will attempt to assign the priority to the hostservice thread.

Return Values	
handle	to a HostService object.
MPIHandleVOID	if the HostService could not be created

See Also

hostServiceDelete | hostServiceEnable | hostServiceStatus

hostService1.c | hostService2.c

hostServiceDelete

Declaration

```
long hostServiceDelete(HostService  service)
```

Required Header: apputil.h

Description

hostServiceDelete alerts the SISR thread to end, polls for the thread to exit, and frees the memory allocated to ***service***.

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	

See Also

hostServiceCreate | hostServiceEnable | hostServiceStatus

hostService1.c | hostService2.c

hostServiceEnable

Declaration

```
long hostServiceEnable(HostService    service,
                       MPI_BOOL       enable)
```

Required Header: apputil.h

Description

hostServiceEnable enables and/or disables the host service SISR thread from executing the ***serviceFunction***. This does not kill the hostService SISR thread. This method is provided to enable or temporarily disable the serviceFunction from servicing host interrupt events.

NOTE: By default, the SISR thread runs with the serviceFunction disabled. This method needs to be called after a hostServiceCreate(...) to enable the execution of the serviceFunction.

If "enabled" is	***Then***
FALSE	HostServiceEnable will disable service.
TRUE	HostServiceEnable will enable service.

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	

See Also

hostServiceCreate | hostServiceDelete | hostServiceStatus

hostService1.c | hostService2.c

hostServiceStatus

Declaration

```
long hostServiceStatus(HostService        service,
                       HostServiceStatus  *status)
```

Required Header: apputil.h

Description

hostServiceStatus returns the status structure for the hostService object.

Return Values	
MPIMessageOK	
MPIMessageARG_INVALID	

See Also

hostServiceCreate | hostServiceDelete | hostServiceEnabled

hostService1.c | hostService2.c

HostServiceFunction

Definition

```
typedef long (MPI_DEF2 *HostServiceFunction)(void *params);
```

Description

Interface prototype for a hostService service function. **HostServiceFunction** can be used to define a custom routine to service sync interrupts. HostServiceFunction function must take a pointer to a structure as a parameter and must return a *long*.

See Also

hostServiceCreate | hostServiceDelete | hostServiceStatus

hostService1.c | hostService2.c

HostServiceStatus

Definition

```
typedef struct HostServiceStatus {
    MPI_BOOL    enabled;
} HostServiceStatus;
```

Description

HostServiceStatus is a structure used to return the status of the hostService object.

	If "enabled" is	*Then*
enabled	FALSE (0)	The hostService SISR function is disabled from servicing host interrupts.
	TRUE (1)	The hostService SISR function is enabled and is servicing host interrupts.

See Also

hostServiceCreate | hostServiceDelete

hostService1.c | hostService2.c