

SynqNet Objects

Introduction

A **SynqNet** object manages a single SynqNet network connected to a motion controller. It represents the physical network. It contains information about the network state, number of nodes, and network status.

A SynqNet network can have one or more [SqNode](#) objects associated with it and each SqNode object can have one or more motors and/or drive interfaces. The SynqNet network provides read/write access to each node. During network initialization, the SynqNet nodes are discovered and mapped to the SynqNet object. The number of motors per SqNode object are determined and mapped to the controller's motor objects.

Methods

Create, Delete, Validate Methods

[meiSynqNetCreate](#)

[meiSynqNetDelete](#)

[meiSynqNetValidate](#)

Configuration and Information Methods

[meiSynqNetCableNumToNodePort](#)

[meiSynqNetConfigGet](#)

[meiSynqNetConfigSet](#)

[meiSynqNetFlashConfigGet](#)

[meiSynqNetFlashConfigSet](#)

[meiSynqNetIdleCableListGet](#)

[meiSynqNetIdleCableStatus](#)

[meiSynqNetInfo](#)

[meiSynqNetPacketFlashConfigGet](#)

[meiSynqNetPacketFlashConfigSet](#)

[meiSynqNetPortToCableNum](#)

[meiSynqNetPacketConfigGet](#)

[meiSynqNetPacketConfigSet](#)

[meiSynqNetStatus](#)

[meiSynqNetTiming](#)

Action Methods

[meiSynqNetFlashTopologyClear](#)

[meiSynqNetFlashTopologySave](#)

[meiSynqNetInit](#)

[meiSynqNetShutdown](#)

Event Methods

[meiSynqNetEventNotifyGet](#)

[meiSynqNetEventNotifySet](#)

[meiSynqNetEventReset](#)

Relational Methods

[meiSynqNetControl](#)

[meiSynqNetNumber](#)

Data Types

[MEISynqNetCableList](#)

[MEISynqNetCableLength](#)

[MEISynqNetCableStatus](#)

[MEISynqNetConfig](#)

[MEISynqNetFailedNodeMask](#)

[MEISynqNetInfo](#)

[MEISynqNetMessage](#)

[MEISynqNetRecoveryMode](#)

[MEISynqNetResourceCommand](#)

[MEISynqNetResourceIoBits](#)

[MEISynqNetResourceMonitor](#)

[MEISynqNetPacketCfg](#)

[MEISynqNetPacketCfgEncoder](#)

[MEISynqNetPacketCfgIo](#)

[MEISynqNetPacketCfgMotor](#)

[MEISynqNetPacketCfgNode](#)

[MEISynqNetPacketCfgProbe](#)

[MEISynqNetResourceCfgProbeDepth](#)

[MEISynqNetState](#)

[MEISynqNetStatus](#)

[MEISynqNetStatusCrcError](#)

[MEISynqNetTiming](#)

[MEISynqNetTrace](#)

Constants

[MEISynqNetMaxCableHOP_COUNT](#)

[MEISynqNetMaxMotorFEEDBACK_PRIMARY_COUNT](#)

[MEISynqNetMaxMotorFEEDBACK_SECONDARY_COUNT](#)

[MEISynqNetMaxMotorCAPTURE_COUNT](#)

[MEISynqNetMaxMotorCOMPARE_COUNT](#)

[MEISynqNetMaxMotorENCODER_COUNT](#)

[MEISynqNetMaxMotorPULSE_ENGINE_COUNT](#)

[MEISynqNetMaxMotors](#)

[MEISynqNetMaxNodeMOTORS](#)

[MEISynqNetMaxNODE_COUNT](#)

[MEISynqNetNodeMaskELEMENTS](#)

[MEISynqNetPacketCfgIo](#)

[MEISynqNetPacketCfgMotor](#)

[MEISynqNetPacketCfgNode](#)

meiSynqNetCreate

Declaration

```
MEISynqNet meiSynqNetCreate(MPIControl control,
                             long number)
```

Required Header: stdmei.h

Description

meiSynqNetCreate creates a SynqNet object identified by ***number***, which is associated with a ***control*** object. SynqNetCreate is the equivalent of a C++ constructor.

control	a handle to a Control object
number	An index to the SynqNet network. First network = 0 Second network = 1 Third network = 2, etc.

Return Values

handle	to a SynqNet object. After creating a SynqNet object it must be validated using <code>meiSynqNetValidate()</code> .
MPIHandleVOID	if the object could not be created

See Also

[meiSynqNetDelete](#) | [meiSynqNetValidate](#)

meiSynqNetDelete

Declaration

```
long meiSynqNetDelete(MEISynqNet synqNet);
```

Required Header: stdmei.h

Description

meiSynqNetDelete deletes a SynqNet object and invalidates its handle. SynqNetDelete is the equivalent of a C++ constructor. All objects that are created must be deleted in the reverse order to avoid memory leaks.

synqNet	a handle to a SynqNet object.
----------------	-------------------------------

Return Values

MPIMessageOK	if <i>SynqNetDelete</i> successfully deleted the object.
---------------------	--

See Also

[meiSynqNetCreate](#) | [meiSynqNetValidate](#)

meiSynqNetValidate

Declaration

```
long meiSynqNetValidate( MEISynqNet  synqNet ) ;
```

Required Header: stdmei.h

Description

meiSynqNetValidate validates the SynqNet object and its handle. SynqNetValidate should be called immediately after an object is created.

synqNet	handle to a valid SynqNet object.
----------------	-----------------------------------

Return Values

MPIMessageOK	if SynqNet is a handle to a valid object.
MEISynqNetMessageINTERFACE_NOT_FOUND	if the controller does not support a SynqNet interface.

See Also

[meiSynqNetCreate](#) | [meiSynqNetDelete](#)

meiSynqNetCableNumToNodePort

Declaration

```
long meiSynqNetCableNumToNodePort ( MEISynqNet      synqNet ,
                                     long              cableNumber ,
                                     long              *nodeNumber ,
                                     MEINetworkPort *port ) ;
```

Required Header: stdmei.h

Description

meiSgNodeConfigGet converts a cable number on a SynqNet network into a node number and port. It reads the node number and writes it into a long pointed to by **nodeNumber** and reads the port and writes it to the value pointed to by **port**.

Network connections can be identified by a cable number OR a node number and port. For simple network topologies, it is easier to identify network connections by a cable number. For complex network topologies, it is easier to identify network connections by a node number and port.

synqNet	a handle to the SynqNet object
cableNumber	the number of the cable
*nodeNumber	a pointer to a node number
*port	a pointer to an enumerated port value

Return Values

MPIMessageOK	if <i>SynqNetCableNumToNodePort</i> successfully converts a cable number on a SynqNet network into a node number and port.
---------------------	--

See Also

[meiSynqNetNodePortToCableNum](#) | [MEISynqNetCableList](#)

meiSynqNetConfigGet

Declaration

```
long meiSynqNetConfigGet(MEISynqNet      synqNet ,
                        MEISynqNetConfig *config);
```

Required Header: stdmei.h

Description

meiSynqNetConfigGet reads a SynqNet network (**synqNet**) configuration and writes it into the structure pointed to by **config**.

synqNet	a handle to a SynqNet object
*config	a pointer to a SynqNet config structure.

Return Values

MPIMessageOK	if <i>meiSynqNetConfigGet</i> successfully reads a SynqNet network's configuration and writes it into the structure pointed to by <i>config</i> .
---------------------	---

See Also

[meiSynqNetInfo](#) | [meiSqNodeConfigGet](#) | [meiSynqNetConfigSet](#)

meiSynqNetConfigSet

Declaration

```
long meiSynqNetConfigSet(MEISynqNet      synqNet ,
                        MEISynqNetConfig *config);
```

Required Header: stdmei.h

Description

meiSynqNetConfigSet writes a SynqNet network (**synqNet**) configuration from the structure pointed to by **config**.

synqNet	a handle to a SynqNet object
*config	a pointer to a SynqNet config structure.

Return Values

MPIMessageOK	if <i>meiSynqNetConfigSet</i> successfully writes a SynqNet network's configuration from the structure pointed to by <i>config</i> .
MPIMessageARG_INVALID	if config is NULL
MPIMessagePARAM_INVALID	if an invalid recoveryMode was specified or a non-zero cable length value was specified for a non-existent cable.
MPISynqNetMessageRING_ONLY	only a value of MEISynqNetRecoveryModeDISABLED for recoveryMode is allowed on a string network.

See Also

[meiSynqNetInfo](#) | [meiSqNodeConfigSet](#) | [meiSynqNetConfigGet](#)

meiSynqNetFlashConfigGet

Declaration

```
long meiSynqNetFlashConfigGet(MEISynqNet      synqNet ,
                               void              *flash ,
                               MEISynqNetConfig *config) ;
```

Required Header: stdmei.h

Description

meiSynqNetFlashConfigGet gets a SynqNet object's flash configuration and writes it in the structure pointed to by **config**.

synqNet	a handle to a SynqNet object
*config	pointer to a locally instantiated MEISynqNetConfig structure.
*flash	<i>flash</i> is either an MEIFlash handle or MPIHandleVOID. If <i>flash</i> is MPIHandleVOID, an MEIFlash object will be created and deleted internally.

Return Values

MPIMessageOK	if <i>meiSynqNetIdleFlashConfigGet</i> successfully writes the Motor's flash configuration to the structure(s).
---------------------	---

See Also

[meiSynqNetFlashConfigSet](#) | [MEISynqNetConfig](#) | [MEIFlash](#)

meiSynqNetFlashConfigSet

Declaration

```
long meiSynqNetFlashConfigSet(MEISynqNet      synqNet ,
                              MEISynqNetConfig *flash ,
                              MEIFlash      *config );
```

Required Header: stdmei.h

Description

meiSynqNetFlashConfigSet gets a SynqNet object's flash configuration and writes it in the structure pointed to by **config**.

NOTE: The network topology must first be saved before changing MEISynqNetConfig.cableLength[n] values in Flash memory. These values will also be cleared when network topology is cleared using meiSynqNetFlashTopologyClear(.).

synqNet	a handle to a SynqNet object
*flash	<i>flash</i> is either an MEIFlash handle or MPIHandleVOID. If <i>flash</i> is MPIHandleVOID, an MEIFlash object will be created and deleted internally. If <i>flash</i> is a valid MEIFlash handle, then the MEIFlash object cache will be updated but the actual write to controller flash will not occur. Use meiFlashMemoryFromFileType(...) to prompt the actual write to flash.
*config	pointer to a locally instantiated MEISynqNetConfig structure that has been initialized by performing a call to meiSynqNetFlashConfigGet(.) or meiSynqNetConfigGet(.) .

Return Values

MPIMessageOK	if <i>meiSynqNetIdleFlashConfigSet</i> successfully sets the Motor's flash configuration using data from the structure(s).
MPIMessageARG_INVALID	if config is NULL.
MPIMessagePARAM_INVALID	if an invalid recoveryMode was specified or a non-zero cable length value was specified for a non-existent cable.

MEISynqNetMessageRING_ONLY	only a value of MEISynqNetRecoveryModeDISABLED for recoveryMode is allowed on a string network.
MEIFlashMessageNETWORK_TOPOLOGY_ERROR	if topology has not yet been saved to flash using meiSynqNetFlashTopologyClear(...)

See Also

[MEISynqNetConfig](#) | [MEIFlash](#) | [meiSynqNetFlashConfigGet](#) | [meiSynqNetConfigGet](#) | [meiSynqNetFlashTopologySave](#) | [meiSynqNetFlashTopologyClear](#)

meiSynqNetIdleCableListGet

Declaration

```
long meiSynqNetIdleCableListGet(MEISynqNet synqNet,
                                MEISynqNetCableList *idleCable);
```

Required Header: stdmei.h

Description

meiSynqNetIdleCableListGet reads a SynqNet network's list of idle cables and writes them into the structure pointed to by **idleCable**. An idle cable has no data traffic. It is the redundant network connection available in ring topologies only. A single ring topology has only one idle cable.

After network initialization, the cable from the last node to the controller is idle by default. If a network fault occurs and the network is configured for fault recovery, the network traffic will be redirected around the faulty connection, through the idle cable, and the faulty connection will become the new idle cable. To test the idle cable, use the `meiSynqNetIdleCableStatus(.)` method.

synqNet	a handle to a SynqNet object
*idleCable	a pointer to a SynqNet cable list structure. The cable list structure contains the number of idle cables and their identifying numbers.

Return Values

MPIMessageOK	if <i>meiSynqNetIdleCableListGet</i> successfully reads a SynqNet network's list of idle cables and writes them into the structure pointed to by <i>idleCable</i> .
---------------------	---

See Also

[meiSynqNetIdleCableStatus](#) | [SynqNet Topologies](#)

meiSynqNetIdleCableStatus

Declaration

```
long meiSynqNetIdleCableStatus(MEISynqNet          synqNet ,
                               long                  cableNumber ,
                               MEISynqNetCableStatus *cableStatus ) ;
```

Required Header: stdmei.h

Description

meiSynqNetIdleCableStatus reads an idle cable's status and writes it into the structure pointed to by ***cableStatus***. Normally, the idle cable has no data traffic. SynqNetIdleCableStatus sends a special test packet across the specified idle cable and then waits for a valid response, then it sends another test packet in the opposite direction and waits for valid response.

The idle cable number for a network can be found using `meiSynqNetIdleCableListGet(.)`. SynqNetIdleCableStatus is not allowed for non-idle cables or when the network is recovering from a fault. During fault recovery (SynqNetState = SYNQ_RECOVERING), the network traffic is redirected around the faulty connection and the idle cable is reassigned. After recovery is complete (SynqNetState = SYNQ), the new idle cable can be tested with SynqNetIdleCableStatus. Use `meiSynqNetStatus(.)` to determine the SynqNet state.

synqNet	a handle to a SynqNet object
cableNumber	the number of the cable to be tested
*cableStatus	a pointer to a SynqNet cable status enumerated value.

Return Values

MPIMessageOK	if <i>meiSynqNetIdleCableStatus</i> successfully reads an idle cable's status and writes it into the structure pointed to by <i>cableStatus</i> .
---------------------	---

See Also

[meiSynqNetIdleCableListGet](#) | [MEISynqNetState](#)

meiSynqNetInfo

Declaration

```
long meiSynqNetInfo(MEISynqNet      synqNet ,
                   MEISynqNetInfo  *info) ;
```

Required Header: stdmei.h

Description

meiSynqNetInfo reads static information about the network associated with the **SynqNet** object and writes it into the structure pointed to by **info**. The SynqNet info structure contains read only data that was determined during network initialization.

synqNet	a handle to a SynqNet object
*info	pointer to a SynqNet network information structure

Return Values

MPIMessageOK	if <i>SynqNetInfo</i> successfully reads the network information and writes it into the info structure.
MPIMessageARG_INVALID	if the info pointer is NULL.

Sample Code

```
/*
   Get the number of nodes on the SynqNet network
   Returns -1 if there is an error
*/
int NumberOfNodes(MEISynqNet synqNet)
{
    MEISynqNetInfo info;
    long returnValue;
    int numNodes = -1;

    returnValue = meiSynqNetInfo(synqNet, &info);
    if(returnValue == MPIMessageOK)
    {
        if(info.nodeCount >= 0 && info.nodeCount <= MEIXmpMaxSynqNetBlocks)
        {
            numNodes = info.nodeCount;
        }
    }
}
```

meiSynqNetInfo

```
}  
return numNodes;  
}
```

See Also

[meiSynqNetIdleCableListGet](#) | [MEISynqNetState](#)

meiSynqNetPacketFlashConfigGet

Declaration

```
long  meiSynqNetPacketFlashConfigGet(MEISynqNet      synqNet ,
                                     void              *flash ,
                                     MEISynqNetPacketCfg *config ) ;
```

Required Header: stdmei.h

Description

meiSynqNetPacketFlashConfigGet is currently unsupported and is reserved for future use.

synqNet	a handle to a SynqNet object
*flash	<i>flash</i> is either an MEIFlash handle or MPIHandleVOID. If <i>flash</i> is MPIHandleVOID, an MEIFlash object will be created and deleted internally. If <i>flash</i> is a valid MEIFlash handle, then the MEIFlash object cache will be updated, but the actual write to controller flash will not occur. Use meiFlashMemoryFromFileType(...) to prompt the actual write to flash.
*config	pointer to configuration structure defined by MEISynqNetPacketCfg .

Return Values

MPIMessageUNSUPPORTED	Currently not supported.
------------------------------	--------------------------

See Also

[meiSynqNetPacketFlashConfigSet](#) | [MEISynqNetPacketCfgNode](#)

meiSynqNetPacketFlashConfigSet

Declaration

```
long  meiSynqNetPacketFlashConfigSet(MEISynqNet      synqNet ,
                                     void             *flash ,
                                     MEISynqNetPacketCfg *config );
```

Required Header: stdmei.h

Description

meiSynqNetPacketFlashConfigSet is currently unsupported and is reserved for future use.

synqNet	a handle to a SynqNet object
*flash	<i>flash</i> is either an MEIFlash handle or MPIHandleVOID. If <i>flash</i> is MPIHandleVOID, an MEIFlash object will be created and deleted internally. If <i>flash</i> is a valid MEIFlash handle, then the MEIFlash object cache will be updated but the actual write to controller flash will not occur. Use meiFlashMemoryFromFileType() to prompt the actual write to flash
*config	pointer to configuration structure defined by MEISynqNetPacketCfg.

Return Values

MPIMessageUNSUPPORTED	Currently not supported.
------------------------------	--------------------------

See Also

[meiSynqNetPacketFlashConfigGet](#) | [MEISynqNetPacketCfgNode](#)

meiSynqNetNodePortToCableNum

Declaration

```
long meiSynqNetNodePortToCableNum( MEISynqNet      synqNet ,
                                   long                nodeNumber ,
                                   MEINetworkPort    port ,
                                   long                *cableNumber ) ;
```

Required Header: stdmei.h

Description

meiSynqNetNodePortToCableNum converts a node number and port on a SynqNet network into a cable number. It reads the cable number and writes it into a long pointed to by ***cableNumber***.

Network connections can be identified by a cable number OR a node number and port. For simple network topologies, it is easier to identify network connections by a cable number. For complex network topologies, it is easier to identify network connections by a node number and port.

synqNet	a handle to the SynqNet object
nodeNumber	the number of the node
port	an enumerated port value
*cableNumber	a pointer to a cable number

Return Values

MPIMessageOK	if <i>SynqNetPortToCableNum</i> successfully converts a node number and port on a SynqNet network into a cable number.
---------------------	--

See Also

[MEISynqNetCableList](#)

meiSynqNetPacketConfigGet

Declaration

```
long meiSynqNetPacketConfigGet(MEISynqNet synqNet,
                               MEISynqNetPacketCfg *config);
```

Required Header: stdmei.h

Description

meiSynqNetPacketConfigGet reads the current network packet configuration for all nodes found on the network to the location pointed to by **config**.

This method is useful for viewing the current network packed data being sent across the network. It is used in conjunction with SynqNetPacketConfigSet. This method is also useful for optimizing network traffic (bandwidth).

Only configurable packet data fields are configured by this method. Fixed packet fields are not application configurable.

synqNet	a handle to a SynqNet object
*config	pointer to configuration structure defined by MEISynqNetPacketCfg.

Return Values

MPIMessageOK	if <i>SynqNetPacketConfigGet</i> successfully reads the network packet configuration.
MPIMessageARG_INVALID	if the config pointer is NULL.
MPIMessageUNSUPPORTED	An sqNodeLib module is missing or it contains a corrupt a resource table.

See Also

[meiSynqNetPacketConfigSet](#) | [MEISynqNetPacketCfgNode](#)

meiSynqNetPacketConfigSet

Declaration

```
long  meiSynqNetPacketConfigSet (MEISynqNet      synqNet ,
                                MEISynqNetPacketCfg *config) ;
```

Required Header: stdmei.h

Description

meiSynqNetPacketConfigSet sets the network packet configuration to the configuration defined in the location pointed to by **config**.

WARNING: meiSynqNetPacketConfigSet(...) is a low-level network routine that will clear other controller object configurations and reset the SynqNet network. This method should be executed in your application before configuring any other MPI objects.

This method is useful for optimizing network traffic (bandwidth).

Only configurable packet data fields are configured by this method. Fixed packet fields are not application configurable.

synqNet	a handle to a SynqNet object
*config	pointer to configuration structure defined by MEISynqNetPacketCfg.

Return Values

MPIMessageOK	if <i>SynqNetPacketConfigSet</i> successfully writes the network packet configuration.
MPIMessageARG_INVALID	if the config pointer is NULL.
MEISynqNetMessageINCOMPLETE_MOTOR	minimum packet requirements for a motor have not been met. Be sure that your motor has at least one command field and one encoder.
MEISynqNetMessageINVALID_AUX_ENC_COUNT	too many secondary encoders have been configured on a node, or there are more secondary encoders than configured/enabled motors.
MEISynqNetMessageINVALID_MOTOR_COUNT	more motors configured than supported by the node.

MEISynqNetMessageINVALID_COMMAND_CFG	an unsupported command value has been selected, or it is not supported by the node.
MEISynqNetMessageINVALID_ENCODER_COUNT	feedback count exceeds MEISynqNetMessageMaxEncoderFEEDBACK_COUNT, or exceeds the count supported by the node.
MEISynqNetMessageINVALID_CAPTURE_COUNT	capture count value is out of range, capture count exceeds capture resources supported by the node, or capture count is non-zero when encoder is disabled (feedbackCount == 0x0).
MEISynqNetMessageINVALID_COMPARE_COUNT	compare count value out of range, compare count exceeds compare resources supported by the node, or compare count is non-zero when encoder is disabled (feedbackCount == 0x0).
MEISynqNetMessageINVALID_INPUT_COUNT	ioInput count is out of range or it exceeds the count supported by the node.
MEISynqNetMessageINVALID_OUTPUT_COUNT	ioOutput count is out of range or it exceeds the count supported by the node.
MEISynqNetMessageINVALID_MONITOR_CFG	monitorConfig is out of range or it exceeds the count supported by the node.
MPIMessageUNSUPPORTED	an sqNodeLib module is missing or it contains a corrupt a resource table.

See Also

[meiSynqNetMessageConfigGet](#) | [MEISynqNetMessageCfgNode](#) | [MEISynqNetMessageCfg](#)

meiSynqNetStatus

Declaration

```
long meiSynqNetStatus(MEISynqNet      synqNet ,
                     MEISynqNetStatus *status) ;
```

Required Header: stdmei.h

Description

meiSynqNetStatus reads status from the network associated with the SynqNet object and writes it into the structure pointed to by status. The SynqNet status structure contains network operation state and error counters. This data is updated every controller sample.

synqNet	a handle to a SynqNet object
*status	pointer to a SynqNet status structure

Return Values

MPIMessageOK	if <i>meiSynqNetStatus</i> successfully gets the network status and writes it into the structure
MPIMessageARG_INVALID	if the status pointer is NULL

See Also

[meiSqNodeStatus](#) | [meiSynqNetInfo](#)

meiSynqNetTiming

Declaration

```
long meiSynqNetTiming(MEISynqNet      synqNet ,
                     MEISynqNetTiming *timing) ;
```

Required Header: stdmei.h

Description

meiSynqNetTiming returns the network timing values to the location pointed to by **timing* for the current operating network. This method can only be called in MEISynqNetStatus.state >= MEISynqNetStateSYNQ.

synqNet	handle to a valid SynqNet object.
*timing	a pointer to a MEISynqNetTiming structure.

Return Values

MPIMessageOK	if <i>meiSynqNetTiming</i> successfully retrieves the network timing values.
MPIMessageARG_INVALID	if <i>*timing</i> equals NULL.

See Also

[MEISynqNetTiming](#) | [MEISynqNetState](#)

meiSynqNetFlashTopologyClear

Declaration

```
long meiSynqNetFlashTopologyClear(MEISynqNet    synqNet ,
                                  MEIFlash      flash ) ;
```

Required Header: stdmei.h

Description

meiSynqNetFlashTopologyClear will clear the topology information that is saved in the controller's flash memory by `meiSynqNetTopologySave()`.

Executing this method will disable the controller from comparing the discovered network topology with the topology saved in flash memory. But, during network initialization, the MPI Library still compares the topology stored in the controller's dynamic memory with the discovered topology. In both cases, the default service commands are sent to the nodes if the network initialization transitions to SYNQ (cyclic) mode.

WARNING: This is a low-level network configuration routine that will clear any SqNode or Motor configurations that use service commands.

The `meiSynqNetFlashTopologyClear` method will:

1. Shutdown the SynqNet network.
2. Reset SqNode and Motor configurations to their defaults. See [MPI Object Configurations that use Service Commands](#).
3. Clear network topology information from Flash and Dynamic memory.
4. Initialize the SynqNet network.

synqNet	a handle to an MEISynqNet object whose network topology is to be cleared.
flash	is either a valid MEIFlash handle or MPIHandleVOID. If flash is MPIHandleVOID, an MEIFlash object will be created and deleted internally.

Return Values**MPIMessageOK**

if *meiSynqNetFlashTopologyClear* successfully clears the networks topology information. *flash* is either an MEIFlash handle or MPIHandleVOID. If *flash* is MPIHandleVOID, an MEIFlash object will be created and deleted internally.

MPISynqNetMessageTOPOLOGY_CLEAR

if topology information has already been cleared or if it has not yet been saved to flash.

See Also

[Save/Clear Topology to Flash](#) | [meiSynqNetFlashTopologySave](#)

meiSynqNetFlashTopologySave

Declaration

```
long meiSynqNetFlashTopologySave( MEISynqNet    synqNet ,
                                  MEIFlash      flash ) ;
```

Required Header: stdmei.h

Description

meiSynqNetFlashTopologySave will save the current SynqNet topology to the controller's flash memory.

After a power-on or network shutdown/init, the controller/MPI Library will compare the discovered network topology with the topology saved in flash memory. If the topology information matches, the controller will automatically transition the network to SYNQ (cyclic) mode and configured service commands are sent to the nodes.

WARNING: This is a low-level network configuration routine that will reset the SynqNet network.

The meiSynqNetFlashTopologySave method will:

1. Shutdown the SynqNet network.
2. Reset SqNode and Motor configurations to their defaults. See [MPI Object Configurations that use Service Commands](#).
3. Clear network topology information from Flash and Dynamic memory.
4. Initialize the SynqNet network.

synqNet	a handle to an MEISynqNet object whose network topology is to be saved.
flash	is either a valid MEIFlash handle or MPIHandleVOID. If flash is MPIHandleVOID, an MEIFlash object will be created and deleted internally.

Return Values**MPIMessageOK**

if *meiSynqNetFlashTopologySave* successfully saves the current SynqNet topology to the controller's flash memory.

MPISynqNetMessageTOPOLOGY_SAVED

if topology information has already been saved to flash. You must clear network topology before saving another network topology to flash.

See Also

[Save/Clear Topology to Flash](#) | [meiSynqNetFlashTopologyClear](#)

meiSynqNetInit

Declaration

```
long meiSynqNetInit(MEISynqNet synqNet);
```

Required Header: stdmei.h

Description

meiSynqNetInit initializes a SynqNet network. This method performs the same network initialization that is automatically done with `mpiControllInit(.)`.

synqNet	a handle to a SynqNet object
----------------	------------------------------

Return Values

MPIMessageOK	if <i>SynqNetInit</i> successfully initializes the network
MPIMessageARG_INVALID	if the <i>eventMask</i> pointer is NULL.

See Also

[meiSynqNetInfo](#) | [meiSynqNetStatus](#)

meiSynqNetShutdown

Declaration

```
long  meiSynqNetPacketFlashConfigSet(MEISynqNet      synqNet ,
                                     void            *flash,
                                     MEISynqNetPacketCfg *config) ;
```

Required Header: stdmei.h

Description

meiSynqNetShutdown disables a SynqNet network by shutting off cyclic data transmission from the controller to the nodes. This will cause the network state to transition to MEISynqNetStateDISCOVERY and nodes will go into a SynqLost state and disable their outputs.

synqNet	a handle to a SynqNet object
----------------	------------------------------

Return Values

MPIMessageOK	if <i>meiSynqNetShutdown</i> successfully disables a SynqNet network by shutting off cyclic data transmission from the controller to the nodes.
---------------------	---

See Also

[meiSynqNetInit](#) | [mpiControlReset](#) | [mpiControlInit](#) | [meiSynqNetStatus](#)

meiSynqNetEventNotifyGet

Declaration

```
long meiSynqNetEventNotifyGet ( MEISynqNet      synqNet ,
                               MPIEventMask   *eventMask ,
                               void                *external ) ;
```

Required Header: stdmei.h

Description

meiSynqNetEventNotifyGet reads the event mask (that specifies the event types for which host notification has been requested) to the location pointed to by `eventMask`, and also writes it into the implementation specific location pointed to by `external`. (if `external` is not NULL).

Use the event mask macros `mpiEventMaskGET(.)`, `mpiEventMaskBitGET(.)`, etc. to decode the `eventMask`.

The event notification data in `external` is in addition to the event notification data in `eventMask`. If either `eventMask` is NULL or `external` is NULL (not both), then the event notification data will not be copied to the NULL pointer.

Remarks

external either points to a structure of type `MEIEventNotifyData{.}` or is NULL.

The `MEIEventNotifyData{.}` structure is an array of controller addresses, whose contents are placed into the `MEIEventStatusInfo{.}` structure (of all events generated by this object).

synqNet	a handle to a SynqNet object
*eventMask	pointer to an event mask, whose bits are defined by the MPI/MEIEventType enumerations.
*external	pointer to external

Return Values

MPIMessageOK	if <i>meiSynqNetEventNotifyGet</i> successfully reads the event mask
MPIMessageARG_INVALID	if the <i>eventMask</i> pointer is NULL.

See Also

[MEI/MPIEventType](#) | [MEIEventNotifyData](#) | [MEIEventStatusInfo](#)

meiSynqNetEventNotifySet

Declaration

```
long meiSynqNetEventNotifySet ( MEISynqNet      synqNet ,
                               MPIEventMask   eventMask ,
                               void              *external ) ;
```

Required Header: stdmei.h

Description

meiSynqNetEventNotifySet requests host notification of the event(s) that are generated by **SynqNet** and specified by **eventMask**, and also specified by the implementation specific location pointed to by **external** (if external is not NULL).

Use the event mask macros `meiEventMaskSYNQNET(.)`, `mpiEventMaskSET(.)`, `mpiEventMaskBitSET(.)`, `mpiEventMaskCLEAR(.)`, etc. to create the eventMask.

The event notification data in **external** is in addition to the event notification data in **eventMask**. If either **eventMask** is NULL or **external** is NULL (not both), then the event notification data will not be copied to the NULL pointer.

Remarks

external either points to a structure of type `MEIEventNotifyData{.}` or is NULL.

The `MEIEventNotifyData{.}` structure is an array of controller addresses, whose contents are placed into the `MEIEventStatusInfo{.}` structure (of all events generated by this object).

synqNet	a handle to a SynqNet object
eventMask	pointer to an event mask, whose bits are defined by the MPI/MEIEventType enumerations.
*external	pointer to external

Return Values

MPIMessageOK	if <i>SynqNetEventNotifySet</i> successfully writes the event mask
MPIMessageARG_INVALID	if the eventMask pointer is NULL

See Also

[MEI/MPIEventType](#) | [MEIEventNotifyData](#) | [MEIEventStatusInfo](#)

meiSynqNetEventReset

Declaration

```
long meiSynqNetEventReset( MEISynqNet      synqNet ,
                           MEIEventMask    eventMask ) ;
```

Required Header: stdmei.h

Description

meiSynqNetEventReset resets the status/event bits that are specified in the eventMask and generated by the synqNet object. After a SynqNet event occurs, SynqNetEventReset should be called to reset the latched status/event bits so the specified event(s) can be generated again.

synqNet	a handle to a SynqNet object
eventMask	An array that defines the event mask bits. The array is defined as: <pre>typedef MPIEventMaskELEMENT_TYPE MPIEventMask[MPIEventMaskELEMENTS]</pre> <p>The bits are defined by the MPI/MEIEventType enumerations.</p>

Return Values

MPIMessageOK

if *meiSynqNetEventReset* successfully resets the status/event bits that are specified in the eventMask and generated by the synqNet object.

See Also

[meiSynqNetStatus](#) | [meiSqNodeEventReset](#) | [meiSqNodeStatus](#)

meiSynqNetControl

Declaration

```
MPIControl meiSynqNetControl(MEISynqNet synqNet);
```

Required Header: stdmei.h

Description

meiSynqNetControl returns a handle to the control object associated with the *SynqNet* object.

synqNet	a handle to a SynqNet object
----------------	------------------------------

Return Values

MPIControl	a handle to a control object.
-------------------	-------------------------------

MPIHandleVOID	if synqNet is not valid.
----------------------	--------------------------

See Also

[meiSynqNetCreate](#) | [mpiControlCreate](#)

meiSynqNetNumber

Declaration

```
long meiSynqNetNumber( MEISynqNet    synqNet ,
                       long            *number ) ;
```

Required Header: stdmei.h

Description

meiSynqNetNumber reads the index of a SynqNet network and writes it into the contents of a long pointed to by number. Each SynqNet network associated with a controller is indexed by an identification number (0, 1, 2, etc.).

synqNet	a handle to a SynqNet object.
*number	a pointer to the index of a SynqNet network.

Return Values

MPIMessageOK	if <i>SynqNetNumber</i> successfully reads the network number.
---------------------	--

See Also

[meiSynqNetInfo](#)

MEISynqNetCableList

Definition

```
typedef struct MEISynqNetCableList {
    long    count;
    long    cableNumber[MEISynqNetCableHOP_COUNT];
} MEISynqNetCableList;
```

Description

MEISynqNetCableList contains the number of cables in the SynqNet network and their identifying numbers. The idle cables can be identified with the `meiSynqNetIdleCableListGet(.)` method.

A cable can also be identified by the node number and port. Use [meiSynqNetCableNumToNodePort\(...\)](#) to translate the cable number to a node number and a port.

count	The number of cables and number of valid elements in the <code>cableNumber[]</code> array.. Range is 0 to <code>MEISynqNetCableHOP_COUNT</code> .
cableNumber	The cables identified by their number. Cables are numbered in the order they are discovered during network initialization.

See Also

[meiSynqNetIdleCableListGet](#) | [meiSynqNetIdleCableStatus](#) | [meiSynqNetCableNumToNodePort](#)

MEISynqNetCableLength

Definition

```
typedef struct MEISynqNetCableLength { /* lengths in meters */
    long minimum;
    long nominal;
    long maximum;
} MEISynqNetCableLength;
```

Description

MEISynqNetCableLength contains the nominal (average), minimum and maximum length in meters for a given network cable.

To disable the cable length checking feature, set the **minimum**, **nominal**, and **maximum** values to zero.

minimum	<p>Minimum cable length in meters.</p> <p>A value less than zero or greater than the nominal value is invalid.</p>
nominal	<p>Nominal cable length in meters.</p> <p>A value less than zero is invalid.</p>
maximum	<p>Maximum cable length in meters.</p> <p>A value less than the nominal value is invalid.</p>

See Also

[MEISynqNetConfig](#) | [meiSynqNetConfigSet](#) | [meiSynqNetConfigSet](#) | [meiSynqNetInfo](#)

MEISynqNetCableStatus

Definition

```
typedef enum MEISynqNetCableStatus {
    MEISynqNetCableStatusGOOD,
    MEISynqNetCableStatusBAD_UPSTREAM,
    MEISynqNetCableStatusBAD_DOWNSTREAM,
    MEISynqNetCableStatusBAD,
} MEISynqNetCableStatus;
```

Description

MEISynqNetCableStatus is an enumeration of a network connection's operating condition. Data transmission is verified in both directions. If the network hardware does not support separate upstream and downstream data path verification, it will report either GOOD or BAD.

MEISynqNetCableStatusGOOD	Network communication across the cable is working properly in both directions.
MEISynqNetCableStatusBAD_UPSTREAM	Network communication across the cable failed in the upstream direction (from node to controller).
MEISynqNetCableStatusBAD_DOWNSTREAM	Network communication across the cable failed in the downstream direction (from the controller to the node).
MEISynqNetCableStatusBAD	Network communication across the cable failed in both directions.

See Also

[meiSynqNetIdleCableStatus](#) | [meiSynqNetIdleCableListGet](#)

MEISynqNetConfig

Definition

```
typedef struct MEISynqNetConfig {
    MEISynqNetRecoveryMode    recoveryMode;
    MEISynqNetCableLength    cableLength[MEISynqNetCableHOP\_COUNT];
} MEISynqNetConfig;
```

Description

MEISynqNetConfig contains configurations for the network's fault recovery mode and the network's cable lengths. The SynqNet configuration can be read with `meiSynqNetConfigGet(.)` and can be written with `meiSynqNetConfigSet(.)`.

recoveryMode	A enumerated value representing the network's fault recovery response mode.
cableLength	<p>An array of cable lengths. Range for cable lengths is 0 to 100 meters.</p> <p>This structure is provided to allow the user application to specify minimum, maximum, and nominal values to use network scheduling equations. Setting a minimum and maximum value will also enable cable length checking at network initialization time. If measured cable lengths fall outside the range defined by the minimum and maximum values, network initialization will fail with a return value of <code>MEISynqNetMessageCABLE_LENGTH_MISMATCH</code>.</p> <p>By default, the values in this structure are zero. Most applications will not need to modify these defaults.</p> <p>NOTE: The network topology must be saved before changing these values in Flash memory. These values will also be cleared when network topology is cleared using <code>meiSynqNetFlashTopologyClear(.)</code></p>

See Also

[meiSynqNetConfigGet](#) | [meiSynqNetConfigSet](#) | [meiSynqNetInfo](#) | [meiSynqNetFlashConfigGet](#) | [meiSynqNetFlashConfigSet](#) | [meiSynqNetFlashTopologyClear](#)

MEISynqNetFailedNodeMask

Definition

```
#define MEISynqNetNodeMaskELEMENTS    (1)

typedef long MEISynqNetFailedNodeMask  [MEISynqNetNodeMaskELEMENTS];
```

Description

MEISynqNetFailedNodeMask is an array of longs with a length of MEISynqNetNodeMaskELEMENTS. Each bit in the failed node mask represents a failed node (0x1 = node 0, 0x2 = node 1, 0x4 = node 2, 0x8 = node 3, etc.). A node failure occurs when the packet error rate counters exceed the packet error rate fail limit.

See Also

[MEISynqNetStatus](#) | [meiSynqNetStatus](#) | [meiSqNodeStatus](#) | [MEISqNodeConfig](#)

MEISynqNetInfo

Definition

```
typedef struct MEISynqNetInfo {
    MEINetworkType      networkType;
    long                nodeCount;
    long                nodeOffset;
    MEISynqNetCableLength cableLength[MEISynqNetCableHOP\_COUNT];
                                /* measured length */
} MEISynqNetInfo;
```

Description

MEISynqNetInfo contains static data that is determined during network initialization. It identifies the network type, number of nodes on the network, starting number (offset) of the first node, and an rough estimate of measured cable lengths.

networkType	contains currently discovered network topology type.
nodeCount	Contains the number of nodes currently discovered on the network.
nodeOffset	Starting node number for nodes on this network. Nodes are currently numbered sequentially across all networks on a controller.
cableLength	<p>An array of measured network cable lengths. These values are estimated values.</p> <p>At network discovery, where network topology has not been saved to flash, the controller will send network packets to measure each cable length and automatically fill out these values for each cable found.</p> <p>These values are estimated values and are the values used in the network scheduling calculations.</p>

Sample Code

```
/*
   Get the number of nodes on the SynqNet network
   Returns -1 if there is an error
*/
int NumberOfNodes(MEISynqNet synqNet)
{
    MEISynqNetInfo info;
    long returnValue;
    int numNodes = -1;

    returnValue = meiSynqNetInfo(synqNet, &info);

    if(returnValue == MPIMessageOK)
    {
        if(info.nodeCount >= 0 && info.nodeCount <= MEIXmpMaxSynqNetBlocks)
        {
            numNodes = info.nodeCount;
        }
    }

    return numNodes;
}
```

See Also

[meiSynqNetInfo](#) | [MEISynqNetConfig](#)

MEISynqNetMessage

Definition

```
typedef enum {  
    MEISynqNetMessageSYNQNET_INVALID,  
    MEISynqNetMessageMAX_NODE_ERROR,  
    MEISynqNetMessageSTATE_ERROR,  
  
    MEISynqNetMessageCOMM_ERROR,  
    MEISynqNetMessageCOMM_ERROR_CRC,  
    MEISynqNetMessageCOMM_ERROR_RX,  
    MEISynqNetMessageCOMM_ERROR_RX_LEN,  
    MEISynqNetMessageCOMM_ERROR_RX_FIFO,  
    MEISynqNetMessageCOMM_ERROR_RX_DRIBBLE,  
    MEISynqNetMessageCOMM_ERROR_RX_CRC,  
  
    MEISynqNetMessageINTERFACE_NOT_FOUND,  
    MEISynqNetMessageTOPOLOGY_MISMATCH,  
    MEISynqNetMessageTOPOLOGY_MISMATCH_FLASH,  
  
    MEISynqNetMessageRESET_REQ_TIMEOUT,  
    MEISynqNetMessageRESET_ACK_TIMEOUT,  
    MEISynqNetMessageDISCOVERY_TIMEOUT,  
    MEISynqNetMessageNO_NODES_FOUND,  
    MEISynqNetMessageNO_TIMING_DATA_AVAIL,  
  
    MEISynqNetMessageINTERNAL_BUFFER_OVERFLOW,  
    MEISynqNetMessageINVALID_MOTOR_COUNT,  
    MEISynqNetMessageINVALID_AUX_ENC_COUNT,  
    MEISynqNetMessageINCOMPLETE_MOTOR,  
    MEISynqNetMessageINVALID_COMMAND_CFG,  
    MEISynqNetMessageINVALID_ENCODER_COUNT,  
    MEISynqNetMessageINVALID_CAPTURE_COUNT,  
    MEISynqNetMessageINVALID_COMPARE_COUNT,  
    MEISynqNetMessageINVALID_INPUT_COUNT,  
    MEISynqNetMessageINVALID_OUTPUT_COUNT,  
    MEISynqNetMessageINVALID_MONITOR_CFG,  
    MEISynqNetMessageINVALID_ANALOG_IN_COUNT,  
    MEISynqNetMessageINVALID_DIGITAL_IN_COUNT,  
    MEISynqNetMessageINVALID_DIGITAL_OUT_COUNT,  
    MEISynqNetMessageINVALID_ANALOG_OUT_COUNT  
  
    MEISynqNetMessageLINK_NOT_IDLE,  
    MEISynqNetMessageIDLE_LINK_UNKNOWN,  
    MEISynqNetMessageRING_ONLY,  
    MEISynqNetMessageRECOVERING,  
  
    MEISynqNetMessageCABLE_LENGTH_UNSUPPORTED,  
    MEISynqNetMessageCABLE_LENGTH_MISMATCH,  
}
```

```

MEISynqNetMessageCABLE_LENGTH_INVALID_NOMINAL,
MEISynqNetMessageCABLE_LENGTH_INVALID_MIN,
MEISynqNetMessageCABLE_LENGTH_INVALID_MAX,

MEISynqNetMessageNODE_FPGA_VERSION,
MEISynqNetMessageMAX_MOTOR_ERROR,
MEISynqNetMessagePLL_ERROR,
MEISynqNetMessageNODE_INIT_FAIL,

MEISynqNetMessageTOPOLOGY_CLEAR,
MEISynqNetMessageTOPOLOGY_SAVED,
MEISynqNetMessageTOPOLOGY_AMPS_ENABLED,

MEISynqNetMessageNODE_MAC_VERSION,
MEISynqNetMessageADC_SAMPLE_FAILURE,

MEISynqNetMessageSCHEDULING_ERROR,

MEISynqNetMessageINVALID_PROBE_CFG,
MEISynqNetMessageINVALID_PROBE_DEPTH,
} MEISynqNetMessage;

```

Required Header: stdmei.h

Change History: Modified in the 03.02.00

Description

MEISynqNetMessage is an enumeration of SynqNet error messages that can be returned by the MPI library.

MEISynqNetMessageSYNQNET_INVALID

The SynqNet number is out of range. This message code is returned by [meiSynqNetCreate\(.\)](#) if the SynqNet network number is less than zero or greater than or equal to MEIXmpMaxSynqNets.

MEISynqNetMessageMAX_NODE_ERROR

The SynqNet node number is out of range. This message code is returned by a SynqNet method if the specified node number is greater than or equal to MEIXmpMaxSynqNetBlocks.

MEISynqNetMessageSTATE_ERROR

The SynqNet network state is not valid. This message code is returned by any method that initializes a SynqNet network if the controller's network state is not a member of the MEIXmpSynqNetState or MEIXmpSynqNetInternalState enumeration. The most commonly used methods that initialize the SynqNet network are: [mpiControlInit\(.\)](#), [mpiControlReset\(.\)](#) and [meiSynqNetInit\(.\)](#). This message code indicates a failure in the controller's initialization sequence. To correct this problem, call [mpiControlReset\(.\)](#).

MEISynqNetMessageCOMM_ERROR

The SynqNet network communication failed. This message code is returned by MPI methods that fail a service command transaction due to a network shutdown. This message indicates a node or network cable failure. To correct this problem, check your network wiring and node condition.

MEISynqNetMessageCOMM_ERROR_CRC

The SynqNet network communication failed due to excessive CRC errors. This message code is returned by MPI methods that fail a service command transaction due to a network shutdown. This message indicates a node or network cable failure. To correct this problem, check your network wiring and node condition.

MEISynqNetMessageCOMM_ERROR_RX

The SynqNet network communication failed due to a Rincon receive error. This message code is returned by MPI methods that fail a service command transaction due to a network shutdown. This message indicates a node or network cable failure. To correct this problem, check your network wiring and node condition.

MEISynqNetMessageCOMM_ERROR_RX_LEN

The SynqNet network communication failed due to a Rincon receive length error. This message code is returned by MPI methods that fail a service command transaction due to a network shutdown. This message indicates a node or network cable failure. To correct this problem, check your network wiring and node condition.

MEISynqNetMessageCOMM_ERROR_RX_FIFO

The SynqNet network communication failed due to a Rincon receive buffer error. This message code is returned by MPI methods that fail a service command transaction due to a network shutdown. This message indicates a node or network cable failure. To correct this problem, check your network wiring and node condition.

MEISynqNetMessageCOMM_ERROR_RX_DRIBBLE

The SynqNet network communication failed due to a Rincon receive dribble error. This message code is returned by MPI methods that fail a service command transaction due to a network shutdown. This message indicates a node or network cable failure. To correct this problem, check your network wiring and node condition.

MEISynqNetMessageCOMM_ERROR_RX_CRC

The SynqNet network communication failed due to a Rincon receive CRC error. This message code is returned by MPI methods that fail a service command transaction due to a network shutdown. This message indicates a node or network cable failure. To correct this problem, check your network wiring and node condition.

MEISynqNetMessageINTERFACE_NOT_FOUND

The controller does not support a SynqNet interface. This message code is returned by [meiSynqNetValidate\(.\)](#), [meiSynqNetFlashTopologySave\(.\)](#), [meiSynqNetFlashTopologyClear\(.\)](#), [mpiControlInit\(.\)](#), and [mpiControlReset\(.\)](#) if the controller does not have a SynqNet hardware interface. To correct this problem, use a controller that supports SynqNet.

MEISynqNetMessageTOPOLOGY_MISMATCH

The network topology does not match the expected network topology. This message code is returned by [mpiControlInit\(.\)](#) or [meiSynqNetInit\(.\)](#) if the discovered network topology does not match the controller's expected network topology (stored in dynamic memory). During the first network initialization the controller stores node identification information (manufacturer, product, and unique values) into it's dynamic memory. This message code indicates the number of nodes, the node order, or types of nodes have changed since the initial network initialization. To correct this problem, either change the network topology to the original configuration or clear the controller's memory with [mpiControlReset\(.\)](#).

MEISynqNetMessageTOPOLOGY_MISMATCH_FLASH

The network topology does not match the expected network topology. This message code is returned by [mpiControlInit\(.\)](#) or [meiSynqNetInit\(.\)](#) if the discovered network topology does not match the controller's expected network topology (stored in flash memory). During the first network initialization the controller stores node identification information (manufacturer, product, and unique values) into it's dynamic memory. Later, when [meiSynqNetFlashTopologySave\(.\)](#) is called, the topology information is stored into flash memory. This message indicates the number of nodes, the node order, or types of nodes have changed since the topology information was stored in flash. To correct this problem, either change the network topology to the saved configuration or clear the controller's flash topology with [meiSynqNetFlashTopologyClear\(.\)](#).

MEISynqNetMessageRESET_REQ_TIMEOUT

The network reset request packet exceeded the timeout. This message code is returned by [mpiControlInit\(.\)](#), [mpiControlReset\(.\)](#), or [meiSynqNetInit\(.\)](#) if the reset request packet fails to traverse the network in the allotted time. This message code indicates a node or network cable failure. To correct this problem, check your network wiring and node condition.

MEISynqNetMessageRESET_ACK_TIMEOUT

The network reset complete packet exceeded the timeout. This message code is returned by [mpiControlInit\(.\)](#), [mpiControlInit\(.\)](#), or [meiSynqNetInit\(.\)](#) if the reset complete packet fails to traverse the network in the allotted time. This message code indicates a node or network cable failure. To correct this problem, check your network wiring and node condition.

MEISynqNetMessageDISCOVERY_TIMEOUT

The network topology discovery exceeded the timeout. This message code is returned by [mpiControlInit\(.\)](#), [mpiControlInit\(.\)](#), or [meiSynqNetInit\(.\)](#) if the controller failed to discover the network topology in the allotted time. This message code indicates a node or network cable failure. To correct this problem, check your network wiring and node condition.

MEISynqNetMessageNO_NODES_FOUND

The controller did not find network nodes. This message code is returned by [mpiControlInit\(.\)](#), [meiSynqNetPacketConfigSet\(.\)](#), or [meiSynqNetInit\(.\)](#) if the controller failed to discover any nodes during network initialization. This message code indicates the first node has failed or the network connection from the controller to the first node is faulty. To correct this problem, check your network wiring and node condition.

MEISynqNetMessageNO_TIMING_DATA_AVAIL

The corresponding SynqNet node module does not contain timing data, so the network cannot be initialized. Contact MEI or drive manufacturer for an updated node module.

MEISynqNetMessageINTERNAL_BUFFER_OVERFLOW

The controller's SynqNet buffer size was exceeded. This message code is returned by [mpiControlInit\(.\)](#) or [mpiControlReset\(.\)](#) if the controller's SynqNet buffer could not be allocated due to overflow. This message code indicates the controller does not have enough memory to initialize the network topology. To correct the problem, either reduce the network nodes or use a different controller model.

MEISynqNetMessageINVALID_MOTOR_COUNT

This message is returned by [meiSynqNetPacketConfigSet\(.\)](#), when the packet configuration contains more motors than supported by the node.

MEISynqNetMessageINVALID_AUX_ENC_COUNT

This message is returned by [meiSynqNetPacketConfigSet\(.\)](#), when the packet configuration contains more secondary encoders than supported by the node.

MEISynqNetMessageINCOMPLETE_MOTOR

This message is returned by [meiSynqNetPacketConfigSet\(.\)](#), when the packet configuration contains a motor configuration that is missing feedback or command fields.

MEISynqNetMessageINVALID_COMMAND_CFG

This message is returned by [meiSynqNetPacketConfigSet\(.\)](#), when the packet configuration contains a different command configuration than is supported by the node.

MEISynqNetMessageINVALID_ENCODER_COUNT

This message is returned by [meiSynqNetPacketConfigSet\(.\)](#), when the packet configuration contains more encoders for feedback than are supported by the node.

MEISynqNetMessageINVALID_CAPTURE_COUNT

This message is returned by [meiSynqNetPacketConfigSet\(.\)](#), when the packet configuration contains a larger capture count than is supported by the node.

MEISynqNetMessageINVALID_COMPARE_COUNT

This message is returned by [meiSynqNetPacketConfigSet\(.\)](#), when the packet configuration contains a larger compare count than is supported by the node.

MEISynqNetMessageINVALID_INPUT_COUNT

This message is returned by [meiSynqNetPacketConfigSet\(.\)](#), when the packet configuration contains a larger ioInput count than is supported by the node.

MEISynqNetMessageINVALID_OUTPUT_COUNT

This message is returned by [meiSynqNetMessageConfigSet\(.\)](#), when the packet configuration contains a larger ioOutput count than is supported by the node.

MEISynqNetMessageINVALID_MONITOR_CFG

This message is returned by [meiSynqNetMessageConfigSet\(.\)](#), when the packet configuration contains more monitor fields than the node can support.

MEISynqNetMessageINVALID_ANALOG_IN_COUNT

This message is returned by [meiSynqNetMessageConfigSet\(.\)](#), when the packet configuration contains a larger analogIn count than is supported by the node.

MEISynqNetMessageINVALID_DIGITAL_IN_COUNT

This message is returned by [meiSynqNetMessageConfigSet\(.\)](#), when the packet configuration contains a larger digitalIn count than is supported by the node.

MEISynqNetMessageINVALID_DIGITAL_OUT_COUNT

This message is returned by [meiSynqNetMessageConfigSet\(.\)](#), when the packet configuration contains a larger digitalOut count than is supported by the node.

MEISynqNetMessageINVALID_ANALOG_OUT_COUNT

This message is returned by [meiSynqNetMessageConfigSet\(.\)](#), when the packet configuration contains a larger analogOut count than is supported by the node.

MEISynqNetMessageLINK_NOT_IDLE

This message is returned by [meiSynqNetMessageIdleCableStatus\(...\)](#) when the cable number supplied is not the idle link. The status check can only be run on an idle cable. See [meiSynqNetMessageIdleCableListGet\(.\)](#).

MEISynqNetMessageIDLE_LINK_UNKNOWN

This message is returned by [meiSynqNetMessageIdleCableListGet\(.\)](#) when an idle cable number in a ring topology cannot be determined. This is due to one or more failed nodes on a ring topology. Be sure to check the status of the nodes.

MEISynqNetMessageRING_ONLY

This message is returned by [meiSynqNetMessageIdleCableListGet\(.\)](#) and [meiSynqNetMessageIdleCableStatus\(...\)](#) because idle cables exist on ring topologies only. It is also returned by [meiSynqNetMessageConfigSet\(...\)](#) when attempting to enable SynqNet recovery mode, which is only supported for ring topologies.

MEISynqNetMessageRECOVERING

The network is recovering from a fault condition. This message is returned by [meiSynqNetIdleCableStatus\(...\)](#) during network fault recovery, because the network traffic is being redirected around the faulted cable and a new idle cable is in the process of being assigned. If you receive this message, wait for recovery to complete and then check the identity of the idle cable with [meiSynqNetIdleCableListGet\(.\)](#).

MEISynqNetMessageCABLE_LENGTH_UNSUPPORTED

This message is returned when a cable on the network is too long.

MEISynqNetMessageCABLE_LENGTH_MISMATCH

This message is returned at network initialization when topology has been saved to flash and a cable length is detected that lies outside the `MEISynqNetConfig.cableLength[n].minimum` and `maximum`. Check cable length (if necessary) and save new cable length values to flash.

MEISynqNetMessageCABLE_LENGTH_INVALID_NOMINAL

The nominal cable length is too long.

MEISynqNetMessageCABLE_LENGTH_INVALID_MIN

The minimum cable length is too long or exceeds nominal value.

MEISynqNetMessageCABLE_LENGTH_INVALID_MAX

The maximum cable length is too long or is less than nominal value.

MEISynqNetMessageNODE_FPGA_VERSION

The node resource FPGA image is out of date. This is only a warning message. Your network will still reach cyclic (SYNQ) mode, but certain node resources may not be available or may not operate as expected. Only ignore this warning if you are aware of FPGA changes between this version and the version built with the MPI or use the `sqNodeFlash.exe` utility to load the most current FPGA version to the node.

MEISynqNetMessageMAX_MOTOR_ERROR

The number of motors on the network exceeds the number set by [MEISynqNetMaxMOTORS](#).

MEISynqNetMessagePLL_ERROR

The node PLL is unable to lock with drive.

MEISynqNetMessageNODE_INIT_FAIL

A node specific initialization routine was unable to complete successfully. Verify node FPGA is the default version for your MPI version. See [MEISqNodeInfoFpga.defaultVersion](#).

MEISynqNetMessageTOPOLOGY_CLEAR

An attempt to clear the saved network topology was made when no network topology has been saved.

MEISynqNetMessageTOPOLOGY_SAVED

An attempt to save network topology was made when the network topology has already been saved. Clear the network topology before attempting to save another topology to flash.

MEISynqNetMessageTOPOLOGY_AMPS_ENABLED

An [meiSynqNetFlashTopologySave /Clear](#) routine has been called while one or more motor amplifiers are enabled. Disables all motor amplifiers ([mpiMotorAmpEnableSet](#)) before calling these low-level routines. To avoid this error message, disable the motor(s) amp enable before calling [meiSynqNetFlashTopologySave/Clear](#).

MEISynqNetMessageNODE_MAC_VERSION

The node MAC FPGA image is out of date. This is an error message. Your network will probably never reach cyclic (SYNQ) mode. Use the `sqNodeFlash.exe` utility to load the most current FPGA version to the node.

MEISynqNetMessageADC_SAMPLE_FAILURE

A check to verify that all analog inputs can be sampled during the given controller sample rate has failed. Either reduce the number of analog inputs on your network (see [meiSynqNetPacketConfigGet / meiSynqNetPacketConfigSet](#)) or decrease your controller sample rate (see [mpiControlConfigGet / mpiControlConfigSet](#)).

MEISynqNetMessageSCHEDULING_ERROR

This is a generic return value to indicate that a problem has occurred while calculating the network scheduling. For more specific information about the error, run your application with timing assignment tracing turned on, using the trace mask:

```
0x00100000    SynqNet: Display timing assignments
```

MEISynqNetMessageINVALID_PROBE_CFG

This message is returned by [meiSynqNetPacketConfigSet\(.\)](#), when the [MEISynqNetPacketCfgProbe](#).count contains a larger count than is supported by the node.

MEISynqNetMessageINVALID_PROBE_DEPTH

This message is returned by [meiSynqNetPacketConfigSet\(.\)](#), when the [MEISynqNetPacketCfgProbe](#).depth contains a larger count than is supported by the node.

See Also

MEISynqNetRecoveryMode

Definition

```
typedef enum MEISynqNetRecoveryMode {  
    MEISynqNetRecoveryModeDISABLED,  
    MEISynqNetRecoveryModeSINGLE_SHOT,  
    MEISynqNetRecoveryModeAUTO_ARM,  
} MEISynqNetRecoveryMode;
```

Description

MEISynqNetRecoveryMode is an enumeration of a network's fault recovery mode. Networks with ring topologies can be configured to recover from a fault condition. A fault condition occurs when a packet error rate counter exceeds its packet error rate fault limit. If fault recovery is enabled and a fault occurs, the node hardware and/or controller will detect the location of the fault and switch the direction of network traffic for nodes downstream from the faulted connection. During fault recovery, the network state is MEISynqNetStateSYNQ_RECOVERING. After the upstream and downstream packet error rate counters decrement to zero, the recovery is completed, and the network state returns to MEISynqNetStateSYNQ.

When fault recovery occurs, the controller will generate a MEIEventTypeSYNQNET_RECOVERY status/event. An application should notify the user about the fault and fix the broken cable/hardware as soon as possible. An application can determine the location of the fault using `meiSynqNetIdleCableListGet(...)`. A network with a ring topology has one idle cable, which has no data traffic. The operation of the idle cable can be tested with `meiSynqNetIdleCableStatus(...)`.

WARNING: If the idle cable is broken and a second fault occurs, then fault recovery will not be able to fully recover from the fault. SynqNet will try to recover, but some nodes will be stranded. Presently, SynqNet does not support an event to notify the application if an idle cable fails. In the meantime, an application can periodically poll the idle cable with `meiSynqNetIdleCableStatus(...)` to test the cable.

MEISynqNetRecoveryModeDISABLED	The network will not attempt to recover from a fault condition. This is the default mode for string topologies.
MEISynqNetRecoveryModeSINGLE_SHOT	The network will only attempt to recover from a fault condition one time. A second fault will be ignored.
MEISynqNetRecoveryModeAUTO_ARM	The network will attempt to recover from a fault condition. After the fault recovery is complete, the network will automatically be re-armed to respond to another fault condition. This is the default mode for ring topologies.

See Also

[meiSynqNetConfigGet](#) | [meiSynqNetConfigSet](#) | [meiSynqNetStatus](#) | [meiSynqNetInfo](#)
| [meiSynqNetIdleCableListGet](#) | [meiSynqNetIdleCableStatus](#)

MEISynqNetResourceCommand

Definition

```
typedef enum MEISynqNetResourceCommand {
    MEISynqNetResourceCommandINVALID,
    MEISynqNetResourceCommandNONE,
    MEISynqNetResourceCommandDAC_ONE,
    MEISynqNetResourceCommandDAC_TWO,
    MEISynqNetResourceCommandDEMAND_A,
    MEISynqNetResourceCommandDEMAND_ABC,
} MEISynqNetResourceCommand;
```

Description

MEISynqNetResourceCommand enumeration is a member of the **MEISynqNetPacketCfgMotor** configuration structure. The values are used to configure the number of 32-bit command data fields sent to drive a motor. Valid values range from greater or equal to **MEISynqNetResourceCommandFIRST** and less than **MEISynqNetResourceCommandLast**, but may be further limited by the available resources on the node.

NOTE: A motor is considered incomplete if it has a command value of **NONE**.

MEISynqNetResourceCommandINVALID	Non-vaild command value was detected.
MEISynqNetResourceCommandNONE	No command data will be sent to this motor. NOTE: this disables the motor and will require the application to remove all other resources for this motor.
MEISynqNetResourceCommandDAC_ONE	Configures a 32bit data field of DAC data to besent to the motor.
MEISynqNetResourceCommandDAC_TWO	Configures two 32bit data fields of DAC data to besent to the motor.
MEISynqNetResourceCommandDEMAND_A	Configures one 16bit data field of Demand data to be sent to the motor.
MEISynqNetResourceCommandDEMAND_ABC	Configures three 16bit data fields of Demand data to be sent to the motor.

See Also

[meiSynqNetPacketConfigGet](#) | [meiSynqNetPacketConfigSet](#) |
[MEISynqNetPacketCfgMotor](#) | [MEISynqNetPacketCfg](#) | [MEISynqNetPacketCfgNode](#)

MEISynqNetResourceIoBits

Definition

```
typedef enum MEISynqNetResourceIoBits {
    MEISynqNetResourceIoBitsINVALID,
    MEISynqNetResourceIoBitsNONE,
    MEISynqNetResourceIoBits16,
    MEISynqNetResourceIoBits48,
} MEISynqNetResourceIoBits;
```

Description

The **MEISynqNetResourceIoBits** enumeration is a member of the **MEISynqNetPacketCfgMotor** configuration structure. The values are used to configure the number of I/O data fields sent to a motor. Valid values range from greater or equal to **MEISynqNetResourceIoBitsFIRST** and less than **MEISynqNetResourceIoBitsLAST**, but may be further limited by the available resources on the node. This is a generic enumeration that is used to configure the resource count of different types of motor and node I/O.

NOTE: (for Motor I/O only) Enabled motors always contain one fixed 16bit data field of dedicated I/O. This is not application configurable.

MEISynqNetResourceIoBitsINVALID	Non-valid bit count was detected.
MEISynqNetResourceIoBitsNONE	No I/O bits will be sent/received.
MEISynqNetResourceIoBits16	One data field of 16 I/O bits will be sent/received.
MEISynqNetResourceIoBits48	One data field of 48 I/O bits will be sent/received.

See Also

[meiSynqNetPacketConfigGet](#) | [meiSynqNetPacketConfigSet](#) | [MEISynqNetPacketCfgMotor](#) | [MEISynqNetPacketCfg](#) | [MEISynqNetPacketCfgNode](#)

MEISynqNetResourceMonitor

Definition

```
typedef enum MEISynqNetResourceMonitor {
    MEISynqNetResourceMonitorINVALID,
    MEISynqNetResourceMonitorNONE,
    MEISynqNetResourceMonitorAB,
    MEISynqNetResourceMonitorABCD,
} MEISynqNetResourceMonitor;
```

Description

The **MEISynqNetResourceMonitor** enumeration is a member of the MEISynqNetPacketCfgMotor configuration structure. The values are used to configure the number of Monitor data fields received from a motor. Valid values range from greater or equal to MEISynqNetResourceMonitorFIRST and less than MEISynqNetResourceMonitorLAST, but may be further limited by the available resources on the node.

MEISynqNetResourceMonitorINVALID	Non-valid bit count was detected.
MEISynqNetResourceMonitorNONE	No monitor data fields will be sent/received for this motor.
MEISynqNetResourceMonitorAB	Two 16bit fields of monitor data will be sent/received for this motor.
MEISynqNetResourceMonitorABCD	Four 16bit fields of monitor data will be sent/received for this motor.

See Also

MEISynqNetPacketCfg

Definition

```
typedef struct MEISynqNetPacketCfg {  
    MEISynqNetPacketCfgNode node[MEISynqNetMaxNODE_COUNT];  
} MEISynqNetPacketCfg;
```

Description

MEISynqNetPacketCfg structure is used as a parameter to the `meiSynqNetPacketConfigGet/Set` methods. It contains the network packet configuration for all nodes found on the network. Only configurable packet data is represented in this structure.

NOTE: Fixed packet fields are NOT application configurable.

node

is an array of configurable packet structures for node data. Array indices 0 through `MEISynqNetInfo.nodeCount` are valid, with a maximum number of motors per node being defined by `MEISynqNetMaxNODE_COUNT`.

See Also

[meiSynqNetPacketConfigSet](#) | [meiSynqNetPacketConfigGet](#) |
[MEISynqNetPacketCfgNode](#) | [MEISynqNetPacketCfgMotor](#) |
[MEISynqNetPacketCfgEncoder](#)

MEISynqNetPacketCfgEncoder

Definition

```
typedef struct MEISynqNetPacketCfgEncoder {
    long feedbackCount;    /* per encoder */
    long captureCount;    /* per encoder */
    long compareCount;    /* per encoder */
} MEISynqNetPacketCfgEncoder;
```

Description

MEISynqNetPacketCfgEncoder structure is a member of the **MEISynqNetPacketCfgMotor** configuration structure. It contains the network packet configuration for a single encoder found on a particular motor. Only configurable packet data is represented in this structure. Fixed packet fields are not application configurable.

Setting all structure members to zero (0x0) values will effectively disable the encoder on the motor. Disabling an encoder will NOT affect encoder numbering, however, encoders with higher index numbers must be disabled before using encoders with lower index numbers.

NOTE: A motor is considered incomplete if it has no encoders enabled.

feedbackCount	<p>configures the number of 32-bit feedback data fields to be sent for an encoder. Valid numbers are zero thru MEISynqNetMaxEncoderFEEDBACK_COUNT , but may be further limited by the available resources on the node.</p> <p>NOTE: Setting this count to a value of zero (0x0) will affectively disable the encoder. All other encoder resources must also be set to NONE or zero (0x0).</p>
captureCount	<p>configures the number of 32-bit capture data fields to be sent for an encoder. Valid numbers are zero thru MEISynqNetMaxMotorCAPTURE_COUNT, but may be further limited by the available resources on the node. If feedbackCount is zero (0x0), then this value must also be zero.</p> <p>NOTE: all capture/compare resources on a motor rely on 2 fixed 32bit data fields for command and status information. These fixed fields are currently not configurable.</p>

compareCount

configures the number of 32-bit compare data fields to be sent for an encoder. Valid numbers are zero thru MEISynqNetMaxMotorCOMPARE_COUNT, but may be further limited by the available resources on the node.

If feedbackCount is zero (0x0), then this value must also be zero.

NOTE: all capture/compare resources on a motor rely on 2 fixed 32bit data fields for command and status information. These fixed fields are currently not configurable.

See Also

[meiSynqNetPacketConfigGet](#) | [meiSynqNetPacketConfigSet](#) |
[MEISynqNetPacketCfgMotor](#) | [MEISynqNetPacketCfg](#) | [MEISynqNetPacketCfgNode](#)

MEISynqNetPacketCfgIo

Definition

```
typedef struct MEISynqNetPacketCfgIo {  
    long    digitalInCount ;  
    long    digitalOutCount ;  
    long    analogInCount ;  
    long    analogOutCount ;  
} MEISynqNetPacketCfgIo ;
```

Description

MEISynqNetPacketCfgIo structure configures the network data packets that contain general purpose I/O.

digitalInCount	Selects the number of 32bit words of digital input data to receive over the SynqNet network.
digitalOutCount	Selects the number of 32bit words of digital output data to send over the SynqNet network.
analogInCount	Selects the number of 32bit words of analog input data to receive over the SynqNet network.
analogOutCount	Selects the number of 32bit words of analog output data to send over the SynqNet network.

See Also

[meiSynqNetPacketConfigSet](#) | [meiSynqNetPacketConfigGet](#) | [MEISynqNetPacketCfgNode](#)

MEISynqNetPacketCfgMotor

Definition

```
typedef struct MEISynqNetPacketCfgMotor {
    MEISynqNetResourceCommand    command;
    long                          pulseEngineCount;
    long                          feedbackPrimaryCount;
    long                          captureCount;
    long                          compareCount;
    MEISynqNetResourceIoBits    ioInput;
    MEISynqNetResourceIoBits      ioOutput;
    MEISynqNetResourceMonitor    monitor;
    MEISynqNetPacketCfgProbe     probe;
} MEISynqNetPacketCfgMotor;
```

Description

MEISynqNetPacketCfgMotor structure is a member of the **MEISynqNetPacketCfgNode** configuration structure. It contains the network packet configuration for a single motor found on a particular node. Only configurable packet data is represented in this structure. Fixed packet fields are not application configurable.

NOTE: Setting all structure members to a NONE or Zero (0x0) value will effectively disable the motor on the network. This will cause the controller to renumber the subsequent motors on the network.

command	Selects the command data type and count. Please see MEISynqNetResourceCommand .
pulseEngineCount	Configures the number of pulse engine to be enable for a motor. Valid numbers are zero thru MEISynqNetMaxMotorPULSE_ENGINE_COUNT , but may be further limited by the available resources on the node. Each pulse engine requires a 64-bit upstream data field and a 32-bit downstream data field.
feedbackPrimaryCount	Configures the number of 32-bit feedback data fields to be sent for a motor. Valid numbers are zero thru MEISynqNetMaxEncoderFEEDBACK_COUNT , but may be further limited by the available resources on the node.

captureCount	<p>Configures the number of 32-bit capture data fields to be sent for a motor. Valid numbers are zero thru MEISynqNetMaxMotorCAPTURE_COUNT, but may be further limited by the available resources on the node. If feedbackCount is zero (0x0), then this value must also be zero.</p> <p>NOTE: All capture/compare resources on a motor rely on two fixed 32bit data fields for command and status information. These fixed fields are currently not configurable.</p>
compareCount	<p>Configures the number of 32-bit compare data fields to be sent for a motor. Valid numbers are zero thru MEISynqNetMaxMotorCOMPARE_COUNT, but may be further limited by the available resources on the node.</p> <p>NOTE: all capture/compare resources on a motor rely on two fixed 32bit data fields for command and status information. These fixed fields are currently not configurable.</p>
ioInput	<p>Selects the number of input bit data received from this motor. Please see MEISynqNetResourceIoBits.</p>
ioOutput	<p>Selects the number of output bit data sent to this motor. Please see MEISynqNetResourceIoBits.</p>
monitor	<p>Selects the number of montior data fields received from this motor. Please see MEISynqNetResourceMonitor.</p>
probe	<p>A structure that specifies the probe count and register depth.</p>

See Also

[meiSynqNetPacketConfigGet](#) | [meiSynqNetPacketConfigSet](#) | [MEISynqNetPacketCfg](#)
[MEISynqNetPacketCfgNode](#) | [MEISynqNetResourceCommand](#) |
[MEISynqNetResourceIoBits](#) | [MEISynqNetResourceMonitor](#) |
[MEISynqNetPacketCfgProbe](#)

MEISynqNetPacketCfgNode

Definition

```
typedef struct MEISynqNetPacketCfgNode {
    MEISynqNetPacketCfgMotor    motor[ MEISynqNetMaxNodeMOTORS ]
    MEISynqNetPacketCfgIo      io;
    long                        feedbackSecondaryCount;
} MEISynqNetPacketCfgNode;
```

Description

MEISynqNetPacketCfgNode is a member of the **MEISynqNetPacketCfg** configuration structure. It contains the network packet configuration for all motors found on a particular node. Only configurable packet data is represented in this structure. Fixed packet fields are not application configurable.

motor	An array of configurable packet structures for motor data. Array indices 0 through MEISqNodeInfo.motorCount are valid, with a maximum number of motors per node being defined by MEISynqNetMaxNodeMOTORS .
io	Configures the network data packets general purpose node I/O.
feedbackSecondaryCount	Configures the number of 32-bit secondary feedback data fields to be sent for a node. Valid numbers are zero thru MEISynqNetMaxMotorFEEDBACK_SECONDARY_COUNT , but may be further limited by the available resources on the node.

See Also

[meiSynqNetPacketConfigGet](#) | [meiSynqNetPacketConfigSet](#) | [MEISynqNetPacketCfg](#)

MEISynqNetPacketCfgProbe

Definition

```
typedef struct MEISynqNetPacketCfgProbe {
    long count;
    MEISynqNetResourceProbeDepth depth[MEISynqNetMaxMotorPROBE\_COUNT];
} MEISynqNetPacketCfgProbe;
```

Description

MEISynqNetPacketCfgProbe specifies the network packet configuration for all the Probes found on a particular motor. Only configurable packet data is represented in this structure. Fixed packet fields are not application configurable.

count	the number of Probe engines per motor. Each Probe engine requires cyclic packet data for the status fields.
depth	an array of enumerated values representing the Probe data depth. Each probe engine can support up to 16 register fields for the probe data. The length of the array is specified by the Probe count.

See Also

[MEISynqNetPacketCfgMotor](#) | [MEISynqNetResourceProbeDepth](#) | [MPIProbeStatus](#)

MEISynqNetResourceCfgProbeDepth

Definition

```
typedef enum MEISynqNetResourceProbeDepth {
    MEISynqNetResourceProbeDepthNONE,
    MEISynqNetResourceProbeDepthTWO,
    MEISynqNetResourceProbeDepthFOUR,
    MEISynqNetResourceProbeDepthEIGHT,
    MEISynqNetResourceProbeDepthSIXTEEN,
} MEISynqNetResourceProbeDepth;
```

Description

MEISynqNetResourceCfgProbeDepth is an enumeration of the possible Probe register depths. Each Probe register is 16 bits. The packet data size is 32 bits. Each Probe engine can support up to 16 register fields for the Probe data.

MEISynqNetResourceProbeDepthNONE	zero Probe data registers will be transmitted upstream
MEISynqNetResourceProbeDepthTWO	2 Probe data registers will be transmitted upstream
MEISynqNetResourceProbeDepthFOUR	4 Probe data registers will be transmitted upstream
MEISynqNetResourceProbeDepthEIGHT	8 Probe data registers will be transmitted upstream
MEISynqNetResourceProbeDepthSIXTEEN	16 Probe data registers will be transmitted upstream

See Also

MEISynqNetState

Definition

```
typedef enum MEISynqNetState {
    MEISynqNetStateINVALID,
    MEISynqNetStateDISCOVERY,
    MEISynqNetStateASYNQ,
    MEISynqNetStateSYNQ,
    MEISynqNetStateSYNQ_RECOVERING,
} MEISynqNetState;
```

Description

MEISynqNetState is an enumeration of the SynqNet network states. Each state shows the present operation mode for the network. Network data traffic occurs in two modes: asynchronous and synchronous.

During **asynchronous** communication, the MPI or controller sends one packet to one node and the node responds with one packet. There are no timing restrictions. Asynchronous communication is used during network initialization, reset, discovery, and node binary download.

During **synchronous** communication, the controller sends packets to the nodes and the nodes respond with packets. All data traffic is scheduled in fixed timeslots to avoid collisions. Packet data updates are cyclic and synchronized to the controller's sample rate. The network state can be read with `meiSynqNetStatus(.)`.

MEISynqNetStateDISCOVERY	This is the initial state before the network is initialized. During this phase, the controller checks the network integrity, determines the network topology, resets the nodes, identifies, initializes, and addresses the nodes. Data packets are sent/received asynchronously.
MEISynqNetStateASYNQ	This state is used for off-line operations, like node binary download. Data packets are sent/received asynchronously.
MEISynqNetStateSYNQ	This is the normal operating state. Data packets are sent/received synchronously.

MEISynqNetStateSYNQ_RECOVERING

During this state, a network fault condition was detected and the network is being reconfigured to route data packets around the fault. After the reconfiguration is complete and the packet error rate counters have decremented to zero, the network will return to the SYNQ state.

See Also

[MEISynqNetStatus](#) | [meiSynqNetStatus](#)

MEISynqNetStatus

Definition

```
typedef struct MEISynqNetStatus {
    MEISynqNetState          state;
    long                    topologySaved; /* TRUE/FALSE */
    MEISynqNetStatusCrcError crcError;
    MPIEventMask             eventMask;
    MEISynqNetFailedNodeMask failedNodeMask;
} MEISynqNetStatus;
```

Description

MEISynqNetStatus contains network state information, CRC counters, eventMask, and the failedNodeMask for a SynqNet network. The network status can be read with the `meiSynqNetStatus(.)` method.

state	Present operation mode for the network.
topologySaved	<p>Contains the status of the network topology.</p> <p>FALSE means the topology is dynamically discovered at initialization.</p> <p>TRUE means that the topology is verified against an expected topology at initialization.</p> <p>See Saving Current Topology To Flash for more information.</p>
crcError	CRC error counters for the controller's network ports. The CRC value increments when received data is corrupt. Range is from 0 to 255. Counter saturates at 255.
eventMask	<p>An array that defines the status for the event mask bits. The array is defined as:</p> <pre>typedef MPIEventMaskELEMENT_TYPE MPIEventMask[MPIEventMaskELEMENTS]</pre> <p>The bits are defined by the <code>MPI/MEIEventType</code> enumerations.</p>

failedNodeMask

Array that defines the failed node mask bits. Each bit represents a failed node (0x1 = node 0, 0x2 = node 2, 0x4 = node 3, etc.). The array is defined as:

```
#define MEISynqNetNodeMaskELEMENTS (1)
typedef long MEISynqNetFailedNodeMask
    [MEISynqNetNodeMaskELEMENTS];
```

See Also

[meiSynqNetStatus](#) | [meiSqNodeStatus](#) | [MEISqNodeStatus](#)

MEISynqNetStatusCrcError

Definition

```
typedef struct MEISynqNetStatusCrcError {  
    long port[MEINetworkPortLAST];  
} MEISynqNetStatusCrcError;
```

Description

MEISynqNetStatusCrcError contains the CRC values for each network port on the controller. This structure is read via the `meiSynqNetStatus(.)` method.

port	an array of CRC values, one for each network port.
-------------	--

See Also

[MEINetworkPort](#) | [MEISynqNetStatus](#) | [meiSynqNetStatus](#) | [CRC Error Counters](#)

MEISynqNetTiming

Definition

```
typedef struct MEISynqNetTiming {
    double controllerFreq;      /* kHz */
    double controllerPeriod;   /* uS  */

    long   txTime;              /* %   */
    double calculationLimit;   /* uS  */

    double driveUpdateFreq;    /* kHz */
    double driveUpdatePeriod; /* uS  */

    long   bandwidthUsage;     /* %   */

    struct {
        double calculationTime; /* uS  */
        double calculationSlack; /* uS  */
        double demandLatency;   /* uS  */
        double feedbackLatency; /* uS  */

        double total;           /* uS  */
        double latencySlack;   /* uS  */
    } controlLatency;

    struct {
        double synq;           /* uS  - synq packet + spacing */
        double demand;        /* uS  - sum demand packets + spacing */
        double control;       /* uS  - sum control packets + spacing */

        double total;         /* uS  */
    } downstream;

    struct {
        double feedback;      /* uS  - sum feedback packets + spacing */
        double status;       /* uS  - sum status packets + spacing */

        double total;         /* uS  */
    } upstream;
} MEISynqNetTiming;
```

Description

MEISynqNetTiming contains static data that is determined during network initialization. It identifies timing statistics about the current network. Most values are just reported based on network timing calculations. The values that can be configured by the MPI are stated below as configurable.

controllerFreq	<p>The sample rate calculation of the SynqNet controller board, in kHz. It is also the rate at which SynqNet packets are sent. The controllerFreq must be configured so that the driveUpdateFreq is an integer multiple of the controllerFreq.</p> <p>This value is configurable. See MPIControlConfig.</p>
controllerPeriod	<p>The sample period calculation of the SynqNet controller board, in uS. Derived from controllerFreq.</p>
txTime	<p>The scheduled time for SynqNet packets to be sent, in % (of the controllerPeriod). Defaults to 75%. Must be set to a value greater than the foreground calculation time, and less than 100%.</p> <p>This value is configurable. See MEIControlConfig.</p>
calculationLimit	<p>The maximum allowed foreground calculation time, in uS. Derived from txTime * controllerPeriod.</p>
driveUpdateFreq	<p>The cyclic communication rate of a drive processor on a SynqNet node, in kHz. Typically fixed to 16 kHz, but may vary depending on drive type. May be configurable on some drives. This rate often matches the drive PWM rate.</p>
driveUpdatePeriod	<p>The cyclic communication period of a drive processor on a SynqNet node, in uS. Derived from driveUpdateFreq.</p>
bandwidthUsage	<p>The amount of SynqNet bandwidth used, in %, for this SynqNet configuration. The actual packet payload configured is divided by the maximum available SynqNet bandwidth, for both upstream and downstream packets. The greater of the two is reported.</p>
calculationTime	<p>The foreground calculation time of the controller, in uS.</p>
calculationSlack	<p>The available slack time between the foreground calculation time, and the scheduled txTime, in uS.</p>
demandLatency	<p>The time to send SynqNet demand data downstream from controller to nodes, in uS. Does not include drive demand delays.</p>
feedbackLatency	<p>The time to send SynqNet feedback data upstream from nodes to controller, in uS. Rounded up to make the total SynqNet latency an integer multiple of the driveUpdatePeriod. Does not include drive feedback delays.</p>

latencySlack	The slack time from the configured feedbackLatency (which is rounded up to an integer multiple of the driveUpdatePeriod), to the minimum possible upstream delay, in uS.
controlLatency	The overall SynqNet system control latency, in uS. Always rounded up to an integer multiple of driveUpdatePeriod. Control latency begins when position feedback is sampled on the node, includes upstream packet delays, controller foreground calculation, downstream packet delays, and ends when demands are strobed on the nodes. Note drive feedback and demand delays are NOT included. Control latency can be broken down into the four components listed below: feedbackLatency, calculationTime, calculationSlack, and demandLatency.
synq	Downstream SYNQ packet payload, in uS. Includes spacing between packets.
demand	Downstream DEMAND packet payload, in uS. Includes spacing between packets.
control	Downstream CONTROL packet payload, in uS. Includes spacing between packets.
downstream	Total downstream (controller to nodes) packet payload, in uS. Includes spacing between packets. Breaks down into the following three packet types.
feedback	Upstream FEEDBACK packet payload, in uS. Includes spacing between packets.
status	Upstream STATUS packet payload, in uS. Includes spacing between packets.
upstream	Total upstream (nodes to controller) packet payload, in uS. Includes spacing between packets. Breaks down into the following two packet types.

See Also

[SynqNet Timing Values](#)

MEISynqNetTrace

Definition

```
typedef enum {
    MEISynqNetTraceDYNA_ALLOC = MEISynqNetTraceFIRST << 0,
    MEISynqNetTraceDYNA_FREE = MEISynqNetTraceFIRST << 1,
    MEISynqNetTraceINIT      = MEISynqNetTraceFIRST << 2,
    MEISynqNetTraceRESET     = MEISynqNetTraceFIRST << 3,
    MEISynqNetTraceTIMING    = MEISynqNetTraceFIRST << 4,
    MEISynqNetTraceSERVICE_CMD = MEISynqNetTraceFIRST << 5,
    MEISynqNetTraceFLOWCTRL  = MEISynqNetTraceFIRST << 6,
    MEISynqNetTraceMAP       = MEISynqNetTraceFIRST << 7,
    MEISynqNetTraceUNMAP     = MEISynqNetTraceFIRST << 8,
} MEISynqNetTrace;
```

Description

MEISynqNetTrace is an enumeration of SynqNet trace bits that can be used to enable/disable library trace statement output for the SynqNet object.

MEISynqNetTraceDYNA_ALLOC	Enables trace statements for controller external memory allocation during SynqNet initialization.
MEISynqNetTraceDYNA_FREE	Enables trace statements for controller external memory de-allocation.
MEISynqNetTraceINIT	Enables trace statements for SynqNet network initialization.
MEISynqNetTraceRESET	Enables trace statements for SynqNet network reset.
MEISynqNetTraceTIMING	Enables trace statements for SynqNet network timing calculations.
MEISynqNetTraceSERVICE_CMD	Enables trace statements for SynqNet service command transactions between the controller and SynqNet nodes.
MEISynqNetTraceFLOWCTRL	This trace bit enables trace statements for the SynqNet service command handshake between the controller and SynqNet nodes.
MEISynqNetTraceMAP	This trace bit enables trace statements for the mapping of firmware pointers to the dynamic controller memory.

MEISynqNetTraceUNMAP

This trace bit enables trace statements for the unmapping of firmware pointers from the dynamic controller memory.

See Also

[Trace.exe utility](#)

MEISynqNetMaxCableHOP_COUNT

Definition

```
#define MEISynqNetCableHOP_COUNT (MEISynqNetMaxNODE_COUNT + 1)
```

Description

MEISynqNetMaxCableHOP_COUNT is the maximum number of cables for a SynqNet network. Presently, string and ring topologies are supported. The maximum number of cables is based on the maximum number of nodes, plus one return cable for ring topologies.

See Also

[MEISynqNetCableList](#) | [MEISynqNetConfig](#)

MEISynqNetMaxMotorFEEDBACK_PRIMARY_COUNT

Definition

```
#define MEISynqNetMaxMotorFEEDBACK_PRIMARY_COUNT (1)
```

Description

MEISynqNetMaxMotorFEEDBACK_PRIMARY_COUNT defines the maximum number of primary feedback resources per motor.

NOTE: Feedback count may be further limited by the available resources on the node.

See Also

MEISynqNetMaxMotorFEEDBACK_SECONDARY_COUNT

Definition

```
#define MEISynqNetMaxMotorFEEDBACK_SECONDARY_COUNT  
    (MEISqNodeMaxFEEDBACK_SECONDARY)
```

Description

MEISynqNetMaxMotorFEEDBACK_SECONDARY_COUNT defines the maximum number of secondary feedback resources per motor.

NOTE: Feedback count may be further limited by the available resources on the node.

See Also

MEISynqNetMaxMotorCAPTURE_COUNT

Definition

```
#define MEISynqNetMaxMotorCAPTURE_COUNT (MEIXmpMaxCapturesPerMotor) /* 2 */
```

Description

MEISynqNetMaxMotorCAPTURE_COUNT defines the maximum number of capture resources per motor.

NOTE: Capture count may be further limited by the available resources on the node.

This define should be used instead of the `MEIXmpMaxCapturesPerMotor` definition in `xmp.h`. It is recommended that applications avoid programming to defines or structures in `xmp.h`

See Also

MEISynqNetMaxMotorCOMPARE_COUNT

Definition

```
#define MEISynqNetMaxMotorCOMPARE_COUNT (MEIXmpMaxCapturesPerMotor) /* 2 */
```

Description

MEISynqNetMaxMotorCOMPARE_COUNT defines the maximum number of compare resources per motor.

NOTE: Compare count may be further limited by the available resources on the node.

This define should be used instead of the MEIXmpMaxComparesPerMotor definition in xmp.h. It is recommended that applications avoid programming to defines or structures in xmp.h

See Also

MEISynqNetMaxMotorENCODER_COUNT

Definition

```
#define MEISynqNetMaxMotorENCODER_COUNT (MEIXmpMotorEncoders) /* 2 */
```

Description

MEISynqNetMaxMotorENCODER_COUNT defines the maximum number of encoder resources per motor.

NOTE: The encoder count may be further limited by the available resources on the node.

This define should be used instead of the `MEIXmpMotorEncoders` definition in `xmp.h`. It is recommended that applications avoid programming to defines or structures in `xmp.h`.

See Also

MEISynqNetMaxMotorPULSE_ENGINE_COUNT

Definition

```
#define MEISynqNetMaxMotorPULSE_ENGINE_COUNT (1)
```

Description

MEISynqNetMaxMotorPULSE_ENGINE_COUNT defines the number of pulse engines per motor.

See Also

MEISynqNetMaxMOTORS

Definition

```
#define MEISynqNetMaxMOTORS (MEIXmpMAX_Motors) /* 32 */
```

Description

MEISynqNetMaxMOTORS defines the maximum number of motors supported on a single SynqNet network.

See Also

MEISynqNetMaxNodeMOTORS

Definition

```
#define MEISynqNetMaxNodeMOTORS ( MEISqNodeMaxMOTORS )
```

Description

MEISynqNetMaxNodeMOTORS defines the maximum number of motor objects supported on a single SynqNet node.

See Also

MEISynqNetMaxNODE_COUNT

Definition

```
#define MEISynqNetMaxNODE_COUNT (MEIXmpMaxSynqNetBlocks) /* 32 */
```

Description

MEISynqNetMaxNODE_COUNT defines the maximum number of nodes supported on a single SynqNet network. This define should be used instead of the **MEIXmpMaxSynqNetBlocks** definition in `xmp.h`. It is recommended that applications avoid programming to defines or structures in `xmp.h`

See Also

MEISynqNetNodeMaskELEMENTS

Definition

```
#define MEISynqNetNodeMaskELEMENTS (1)
typedef long MEISynqNetFailedNodeMask [MEISynqNetNodeMaskELEMENTS];
```

Description

MEISynqNetNodeMaskELEMENTS defines the number of data elements in an [MEISynqNetFailedNodeMask](#).

See Also

[MEISynqNetFailedNodeMask](#)