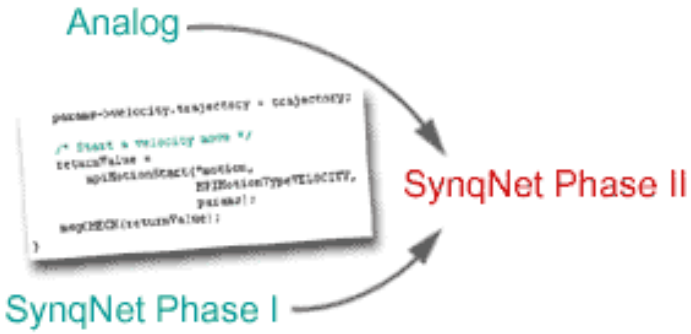
Upgrading Software to SynqNet Phase II

documentation

The introduction of SynqNet Phase II has necessitated changes to motor I/O and capture interfaces in the MPI library. Since the number of available motor I/O pinouts vary from node to node, different structures must be used to enumerate I/O signal access through the MPI. Some of the changes necessary to switch from old versions of the MPI (Analog and SynqNet Phase I) to SynqNet Phase II MPI are listed below.

Contents



Additional Include File and Directory

To access the node-specific enumerations, the file "sqNodeLib.h" must be #included in application code. This requires the addition of the directory, "\xmp\sqNodeLib\include" to the list of additional include directories in the Project->Settings C/C++ tab.

If only general MPI enumerations are used, then there is no need to #include "sqNodeLib.h" in an application.

Motor I/O Enumeration Changes

With old versions of the MPI, all digital inputs were included into one enumeration, MEIMotorInput, while all digital outputs were included in another, MEIMotorOutput.

With SynqNet Phase II, digital I/O has been split up into 3 categories:

- **Dedicated Inputs** - represented by MEIMotorDedicatedIn, refers to common inputs such as the home line.
- **Dedicated Outputs** - represented by MEIMotorDedicatedOut, refers to common outputs such as the brake enable line.
- **Configurable and/or Node** - Specific I/O, represented by MEIMotorIoMask. Presently, these only support digital inputs or outputs. In the future, support for stepper motor output and compares will be supported by these I/O.

When upgrading application code to SynqNet Phase II, check the drive manufacturer's configuration type and documentation for details about which configurable I/O are supported. If the enumerations are replaced without verifying a drive's capabilities, an application will compile, but it may not work.

The tables below shows how to upgrade motor I/O enumerations to SynqNet Phase II.

Analog & SynqNet Phase I MPI		SynqNet Phase II MPI
MEIMotorInputAMP_FAULT	→	MEIMotorDedicatedInAMP_FAULT
MEIMotorInputHOME	→	MEIMotorDedicatedInHOME
MEIMotorInputOVERTRAVEL_POS	→	MEIMotorDedicatedInLIMIT_HW_POS
MEIMotorInputOVERTRAVEL_NEG	→	MEIMotorDedicatedInLIMIT_HW_NEG
MEIMotorInputINDEX	→	MEIMotorDedicatedInINDEX
MEIMotorInputBROKEN_WIRE	→	MEIMotorDedicatedInFEEDBACK_FAULT
MEIMotorInputILLEGAL_STATE	→	
(No equivalent enumeration)	→	MEIMotorDedicatedInBRAKE_APPLIED
(No equivalent enumeration)	→	MEIMotorDedicatedInDRIVE_STATUS_0
(No equivalent enumeration)	→	MEIMotorDedicatedInDRIVE_STATUS_1
(No equivalent enumeration)	→	MEIMotorDedicatedInDRIVE_STATUS_2
MEIMotorInputSIM4_INDEX	→	(No equivalent enumeration) Sinusoidal Interpolation Not Supported
MEIMotorInputSIM4_ENCA		
MEIMotorInputSIM4_ENCB		

(No equivalent enumeration)	→	MEIMotorDedicatedOutAMP_ENABLE
MEIMotorOutputBRAKE_ENABLE	→	MEIMotorDedicatedOutBRAKE_RELEASE

Analog & SynqNet Phase I MPI		RMB Specific Values †	Equivalent General MPI Values
MEIMotorTransceiverConfigINPUT	→	(none - use generic MPI value –>)	MEIMotorIoTypeINPUT
MEIMotorTransceiverConfigOUTPUT	→	(none - use generic MPI value –>)	MEIMotorIoTypeOUTPUT
MEIMotorTransceiverConfigSTEP	→	(No equivalent enum -- Step Engine Not Yet Implemented)	
MEIMotorTransceiverConfigDIR	→	(No equivalent enum -- Step Engine Not Yet Implemented)	
MEIMotorTransceiverConfigCW	→	(No equivalent enum -- Step Engine Not Yet Implemented)	
MEIMotorTransceiverConfigCCW	→	(No equivalent enum -- Step Engine Not Yet Implemented)	
MEIMotorTransceiverConfigQUAD_A	→	(No equivalent enum -- Step Engine Not Yet Implemented)	
MEIMotorTransceiverConfigQUAD_B	→	(No equivalent enum -- Step Engine Not Yet Implemented)	
MEIMotorTransceiverConfigCOMPARE	→	(No equivalent enum -- Step Engine Not Yet Implemented)	
MEIMotorTransceiverIdA	→	RMBMotorIoConfigXCVR_A	MEIMotorIoConfigIndex0
MEIMotorTransceiverIdB	→	RMBMotorIoConfigXCVR_B	MEIMotorIoConfigIndex1
MEIMotorTransceiverIdC	→	RMBMotorIoConfigXCVR_C	MEIMotorIoConfigIndex2
MEIMotorTransceiverExtendedIdD	→	RMBMotorIoConfigXCVR_D	MEIMotorIoConfigIndex3
MEIMotorTransceiverExtendedIdE	→	RMBMotorIoConfigXCVR_E	MEIMotorIoConfigIndex4
MEIMotorTransceiverExtendedIdF	→	RMBMotorIoConfigXCVR_F	MEIMotorIoConfigIndex5
(No equivalent enumeration)	→	RMBMotorIoConfigUSER_0_IN	MEIMotorIoConfigIndex6
(No equivalent enumeration)	→	RMBMotorIoConfigUSER_0_OUT	MEIMotorIoConfigIndex7
MEIMotorInputXCVR_A	→	RMBMotorXcvrA	MEIMotorIoMask0
MEIMotorInputXCVR_B	→	RMBMotorXcvrB	MEIMotorIoMask1
MEIMotorInputXCVR_C	→	RMBMotorXcvrC	MEIMotorIoMask2
MEIMotorInputUSER	→	RMBMotorOptoUSER0_IN	MEIMotorIoMask6
MEIMotorInputUSER_0	→	RMBMotorOptoUSER0_OUT	MEIMotorIoMask6
MEIMotorInputUSER_1	→	(Not Supported by RMB)	(Vendor Specific)

MEIMotorOutputXCVR_A	→	RMBMotorXcvrA	MEIMotorIoMask0
MEIMotorOutputXCVR_B	→	RMBMotorXcvrB	MEIMotorIoMask1
MEIMotorOutputXCVR_C	→	RMBMotorXcvrC	MEIMotorIoMask2
MEIXmpMotorIOMaskXCVR_D	→	RMBMotorXcvrD	MEIMotorIoMask3
MEIXmpMotorIOMaskXCVR_E	→	RMBMotorXcvrE	MEIMotorIoMask4
MEIXmpMotorIOMaskXCVR_F	→	RMBMotorXcvrF	MEIMotorIoMask5
MEIMotorOutputUSER	→	RMBMotorOptoUSER0_OUT	MEIMotorIoMask7
MEIMotorOutputUSER_0	→	RMBMotorOptoUSER0_OUT	MEIMotorIoMask7
MEIMotorOutputUSER_1	→	(Not Supported by RMB)	(Vendor Specific)

† - RMB Specific Values are for only MEI RMB-10V Nodes

Changes to the MEIMotorConfig structure

Analog (20020117.1.6)

```
typedef struct MEIMotorConfig {
  ...
        MEIMotorTransceiver   Transceiver[MEIXmpMotorTransceivers];
        MEIMotorTransceiver   TransceiverExtended[MEIXmpMotorTransceiversExtended];
        long                  UserOutInvert;   /* Output Polarity */
  ...
} MEIMotorConfig;


typedef struct MEIMotorTransceiver {
    long                       Invert;  /* TRUE = invert (not valid for INPUT) */
    MEIMotorTransceiverConfig  Config;
} MEIMotorTransceiver;
```

SynqNet Phase I (20011220.1.16)

```
typedef struct MEIMotorConfig {
  ...
    MEIMotorTransceiver Transceiver[MEIXmpMotorTransceivers];
    MEIMotorTransceiver TransceiverExtended[MEIXmpMotorTransceiversExtended];
    long                UserOutInvert[MEIXmpMotorUserOptos]; /* Output Polarity */
  ...
} MEIMotorConfig;


typedef struct MEIMotorTransceiver {
    long                       Invert;  /* TRUE = invert (not valid for INPUT) */
    MEIMotorTransceiverConfig   Config;
} MEIMotorTransceiver;
```

SynqNet Phase II (20030620.1.1)

```
typedef struct MEIMotorConfig {
  ...
    MEIMotorIoConfig    Io[MEIMotorIoConfigIndexLAST];
  ...
} MEIMotorConfig;



typedef struct MEIMotorIoConfig {
    MEIMotorIoType    Type;

} MEIMotorIoConfig;
```

MEIMotorConfig Parameter Changes

print friendly version

The table below shows how to upgrade motor I/O configuration code to SynqNet Phase II.

Analog & SynqNet Phase I MPI			SynqNet Phase II MPI	
Type	**Parameter**		**Type**	**Parameter**
long	MEIMotorConfig.Transceiver[].Invert	→		(No equivalent enum -- No longer supported)
MEIMotorTransceiverConfig	MEIMotorConfig.Transceiver[].Config	→	MEIMotorIoType	MEIMotorConfig.Io[].Type
long	MEIMotorConfig.TransceiverExtended[].Invert	→	long	No equivalent enum. No longer supported.
MEIMotorTransceiverConfig	MEIMotorConfig.TransceiverExtended[].Config	→	MEIMotorIoType	MEIMotorConfig.Io[].Type
long	MEIMotorConfig.UserOutInvert (analog)	→	long	(No equivalent enum -- No longer supported)
long	MEIMotorConfig.UserOutInvert[0] (phase I)	→	long	(No equivalent enum -- No longer supported)
(all parameters)	MEIMotorConfig.Stepper.Xxx	→	(No equivalent enum -- Step Engine Not Yet Implemented)	
long	MEIMotorConfig.pulseEnable	→	(No equivalent enum -- Compare Engine Not Yet Implemented)	
long	MEIMotorConfig.pulseWidth	→	(No equivalent enum -- Step & Compare Engines Not Yet Implemented)	

New Capture Engine Features

The next few sections on captures explain how to update code for the old capture engine to code for the new capture engine.

The new capture engine includes additional support for "global" capturing (capturing the positions of multiple encoders from a single trigger). Global capturing is only global to a single SynqNet node. The reason for this is because position capturing is supposed to be a high-speed process, on the order of tens of nanoseconds. Communicating information across the SynqNet network would seriously degrade the timing of the capture engine.

The way the global capturing works is that an internal "global" bit is kept inside a node's FPGA. This bit is considered as one of the inputs into a capture's engine. Each capture also has the ability to link this bit to its trigger state. (Please see the Capture Engine Diagrams.) At any given time, only one capture may link to the global bit. One should not configure a capture to use the global bit as an input and then link to the global bit with the same capture.

In SynqNet Phase II MPI, instead of a trigger mask for capture, there is now an array of trigger sources, each with an enable and invert option. The following few sections will explain how this array functions.

The new capture engine supported by SynqNet Phase II MPI, also supports capturing on different edges of the trigger state. (Please see the Capture Engine Diagrams.) Capturing can now happen on the rising edge, the falling edge, or any transition of the trigger state. This is specified by the ***edge*** member of the MPICaptureConfig structure:

MPICaptureEdgeRISING	Capture on the rising edge of the trigger state
MPICaptureEdgeFALLING	Capture on the falling edge of the trigger state
MPICaptureEdgeEITHER	Capture on an transition of the trigger state
MPICaptureEdgeNONE	Never Capture

Capture Enumeration Changes

The table below shows how to upgrade capture enumerations to SynqNet Phase II.

Analog & SynqNet Phase I MPI		SynqNet Phase II MPI
MPIIoType	→	(Enumeration no longer necessary)
MEIMotorInputHOME	→	MPICaptureSourceHOME
MEIMotorInputOVERTRAVEL_POS	→	MPICaptureSourceOVERTRAVEL_POS
MEIMotorInputOVERTRAVEL_NEG	→	MPICaptureSourceOVERTRAVEL_NEG
MEIMotorInputINDEX	→	MPICaptureSourceINDEX
MEIMotorInputXCVR_A	→	MPICaptureSourceMOTOR_IO_0 **(MEI RMB Only)**
MEIMotorInputXCVR_B	→	MPICaptureSourceMOTOR_IO_1 **(MEI RMB Only)**
MEIMotorInputXCVR_C	→	MPICaptureSourceMOTOR_IO_2 **(MEI RMB Only)**
(No equivalent enumeration)	→	MPICaptureSourceGLOBAL
(No equivalent enumeration)	→	MPICaptureSourceINDEX_SECONDARY
(No equivalent enumeration) Configured through MPIMotorConfig.captureOnChange	→	MPICaptureEdge See MPICaptureConfig Parameter Changes section.

Changes to the MEICaptureConfig structure

Analog / SynqNet Phase I (20020117.1.6 / 20011220.1.16)

```
typedef struct MPICaptureConfig {
    MPIIoTrigger    trigger;
    long            latchCount;
  ...
} MPICaptureConfig;



/* MPIIoTrigger */
typedef struct MPIIoTrigger {
  ...
    unsigned long     mask;
    unsigned long     pattern;
} MPIIoTrigger;
```

SynqNet Phase II (20030620.1.1)

```
typedef struct MPICaptureConfig {
    MPICaptureTrigger          source[MPICaptureSourceCOUNT];
    MPICaptureEdge             edge;
    MPICaptureTriggerGlobal    global;
    MPICaptureType             type;
    long                       captureMotorNumber;
    long                       feedbackMotorNumber;
                                  /* the same as captureMotorNumber for POSITION
capture */
    MPIMotorEncoder            encoder;
    long                       captureIndex /* 0, 1,... */
} MPICaptureConfig;



typedef struct MPICaptureTrigger {
    long enabled;   /* TRUE/FALSE */
    long invert;    /* TRUE = invert, FALSE = normal */
} MPICaptureTrigger;


typedef struct MPICaptureTriggerGlobal {
    long enabled;   /* TRUE/FALSE */
} MPICaptureTriggerGlobal;
```

MPICaptureConfig Parameter Changes

print friendly version

The table below shows how to upgrade capture configuration code to SynqNet Phase II.

Trigger when input is:	Analog / SynqNet Phase I		SynqNet Phase II	
	trigger.mask	trigger.pattern	source[sourceBit].enabled	source[sourceBit].invert
High	Include† MPIMotorInput bit	Include† MPIMotorInput bit	TRUE	FALSE
Low	Include† MPIMotorInput bit	Exclude†† MPIMotorInput bit	TRUE	TRUE
Don't use as a trigger	Exclude MPIMotorInput bit	Exclude MPIMotorInput bit	FALSE	(n/a)
Use Global Trigger Flag for Capture	(n/a)		source[MPICaptureSourceGLOBAL].enabled = TRUE	
			source[MPICaptureSourceGLOBAL].invert = FALSE	
Set Global Trigger FlagOn Capture	(n/a)		MPICaptureConfig.global = TRUE	

† - The bit associated with this motor input should be on
†† - The bit associated with this motor input should be off

New MPICaptureConfig Parameter Descriptions

type	Specifies either postion-based or time-based capture. Use MPICaptureTypePOSITION for position-based capture and MPICaptureTypeTIME for time-based capture.
captureMotorNumber	The number of the motor whose "source" (MPICaptureTrigger) is used to capture position.
feedbackMotorNumber	The number of the motor whose position is being returned from the capture event. (It must be the same as captureMotorNumber for position capture).
encoder	Specifies the encoder feedback being captured.
captureIndex	A zero-based index that specifies which capture resource on an axis is to be associated with the capture object. Each axis on a node has a given number of captures associated with it. Normally, each axis has only 1 capture resource, but an azis may have up to 4 capture resources.

Creating MPICapture Objects

The way MPICapture Objects are created has also changed. MPICapture objects are now associated with encoders. Before SynqNet, each capture could easily be enumerated across the entire controller, and thus MPICapture objects were created using an MPIControl handle. But with SynqNet, different drives can support different numbers of encoders, so this enumeration is now impossible. In SynqNet Phase II, MPICapture objects are now created using an MPIMotor handle. SynqNet Phase I had two different ways to access capture objects, one of which was identical to the analog MPI and the second was specific to SynqNet Phase I. The table below lists some of the differences between the ways capture objects are created or accessed.

MPI Version	Method for Creating/Accessing Capture Objects	Handle Type Used to Create Capture Objects	Capture Objects Need Deletion?
Analog	mpiCaptureCreate	MPIControl	Yes
SynqNet Phase I	mpiCaptureCreate	MPIControl	Yes
SynqNet Phase I	meiMotorCapture	MPIMotor	No†
SynqNet Phase II	mpiCaptureCreate	MPIControl	Yes

† - These capture objects were created and deleted by the motor object.

The function prototypes for creating or accessing capture objects is listed below:

Analog / SynqNet Phase I

```
const MPICapture mpiCaptureCreate(MPIControl  control,
                                  long        number);


const MPICapture meiMotorCapture(MPIMotor  motor,
                                 long      index);
```

SynqNet Phase II

```
MPICapture mpiCaptureCreate(MPIControl  control,
                            long        number);
```

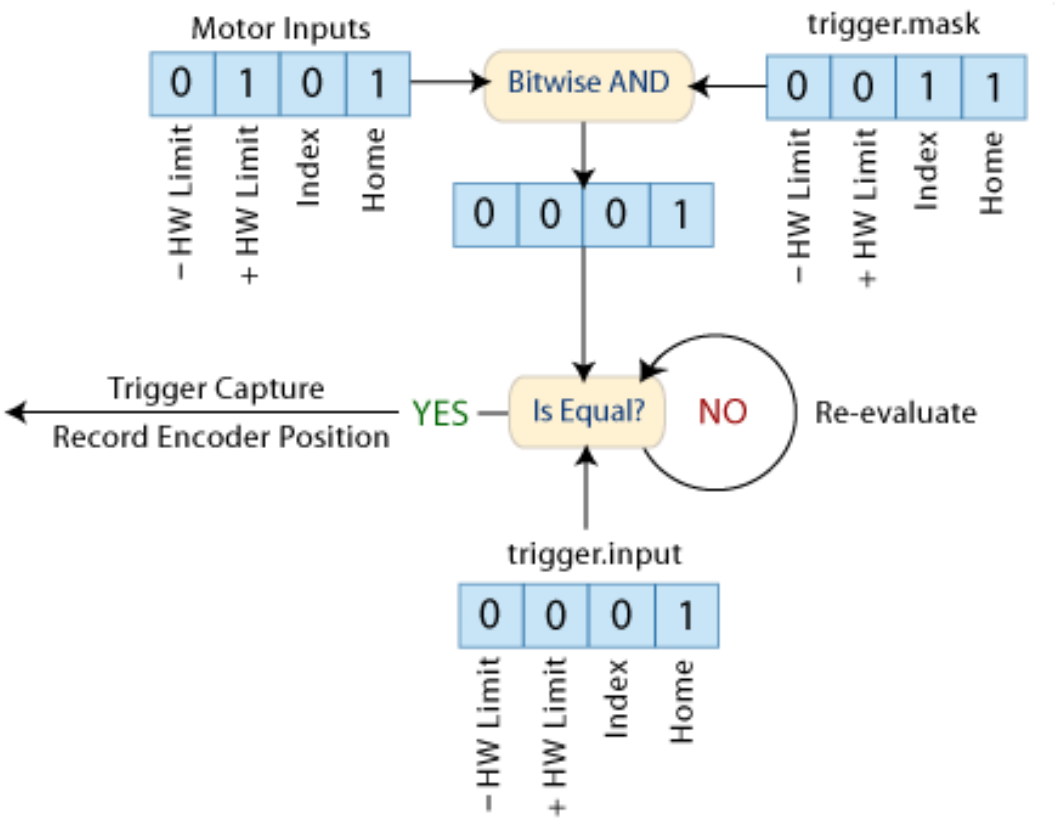
The encoder argument identifies which encoder to create a capture for.

The *number* argument tells which capture to use on a particular encoder. *Number* represents the index of a zero-indexed array of captures, similar to the *number* argument for other mpiCreate… methods.

The function prototype for the for the mpiCaptureDelete() method has not changed.

Capture Engine Diagrams

Analog / SynqNet Phase I Capture Engine



SynqNet Phase II
Capture Engine