SS555-DK

RTL 555

Runtime Libraries for the SS555 Microcontroller

31/10/01

1. Table Of Contents

2. Special Data Types

The RTL555 uses a slightly non-standard nomenclature for their data types. This nomenclature can either be used or ignored by the user. The table below is a list of the RTL555 data type and its C equivalent.

RTL555	Range	C
pInt8		signed char*
Int8	-127 to 127	signed char
pUInt8		unsigned char*
UInt8	0 to 255	unsigned char
PVInt8		volatile signed char*
VInt8	-127 to 127	volatile signed char
PVUInt8		volatile unsigned char*
VUInt8	0 to 255	volatile unsigned char
pInt16		signed short*
Int16	-32,767 to 32,767	signed short
pUInt16		unsigned short*
UInt16	0 to 65,535	unsigned short
PVInt16		volatile signed short*
VInt16	-32,767 to 32,767	volatile signed short
PVUInt16		volatile unsigned short*
VUInt16	0 to 65,535	volatile unsigned short
pInt32	-2,147,483,647 to 2,147,483,647	signed int
Int32		signed int
pUInt32		unsigned int*
UInt32	0 to 4,294,967,295	unsigned int
PVInt32		volatile signed int*
VInt32	-2,147,483,647 to 2,147,483,647	volatile signed int
PVUInt32		volatile unsigned int*
VUInt32	0 to 4,294,967,295	volatile unsigned int
pInt64		signed long long*
Int64		signed long long
pUInt64		unsigned long long*
UInt64		unsigned long long
pVInt64		volatile signed long long*
VInt64		volatile signed long long
pVUInt64		volatile unsigned long long*
VUInt64		volatile unsigned long long
pFloat32		float*
Float32	6 digits of precision	float
pFloat64		double*
Float64	10 digits of precision	double

Table 1 - RTL555 data types and their C equivalents

All the above are implemented through typedefs located in the m_common.h

The 64 bit data types are non-standard gcc nomenclature.

3. ReturnTypes

The functions in the RTL555 that do not return a value are listed as ReturnTypes. Error checking is performed upon all variables passed to a particular function. If a variable lies outside the range of acceptable values, an error code is generated and returned. Using this along with the type checking offered by the GNU tools can be used to find errors in the user's code quickly. The error checking code can then be removed in the final application to reduce program size and increase the program speed.

Using Error Codes

The SS555-SDK has two sets of library files, libintec.h and libdebug.h. These library files are included in every project by adding either –lintec or –ldebug in the include libraries dialog box under program settings. If –ldebug is included, the error checking is used. If –lintec is included, the error checking is omitted.

Reading Error Codes

ReturnTypes consist of a two-digit code that can be used to determine which variable is in error and what type of error occurred. This number is not representative of all errors that occur, but only the variable in the list that is in error.

X	X
The first variable number that is in error	The type of error experienced.

0	Variable is too low
1	Variable is too high
2	Variable, though within range, is not allowed
3	Special Error Code. See the function documentation for more information.

Table 2 - Error Types

Error Code Example

In the example below, the program is trying to write to MPIO 17. The MPIO only has pin numbers from 0 to 15. Errorcheck would be 11 in both cases, the first variable is too large. Should the MPIO pin be less than zero, the compiler will generate a warning. If this warning is not handled, the errorcheck will still generate a 11 as the negative number becomes a very large unsigned positive number.
Once this program correctly works, the error checking can be removed from the final project code by changing the library in use to –lintec instead of –ldebug in the File-Program Settings menu option in IPM.

```
int main( void ){
   ReturnType errorcheck;
   errorcheck = MPIO_pin_direction_set( 17 , OUTPUT );
   printf ("\nError code %d", errorcheck);
   errorcheck = MPIO_pin_set ( 17 , HI );
   printf ("\nError code %d", errorcheck);
   return( 0 );
```

4. List of RTL555 Functions

COL functions	
COL_priority_set(priority , function)*	Set the level to interrupt
COL_isr_set(COL , Error)	Setup the COL interrupt vector table
COL_isr(void)	Execute correct COL isr
COL_interrupt_enable(enable)	Enable/disable the COL interrupt
MIOS (CS/DA/PWM) Interrupt Functions	
MIOS_init(which_MIOS , function)	Set the spurious MIOS ISR.
MIOS_priority_set(which_MIOS,priority,function)	Set the level to interrupt
MIOS_isr_set(which_MIOS,which_pin,function , enable)	Setup and enable a single entry in the MIOS interrupt vector table
MIOS0_isr(void)	Execute correct MIOS0 isr
TB / DEC combined functions	
TMBCLK_set(Clock Source,Enable)	sets combined TB/DEC properties
DEC only functions	
DEC_set(time)	sets the current value of the DEC in µs
DEC_get( )	returns the current value of the DEC
DEC_isr_set(function)	Setup the DEC interrupt vector table
Flash Programming	
Flash_enable(enable)	Enables/disables flash programming on the SS555. Sets correct levels on both VPP and EPEE.
Interrupt Controller	
defaultISR_1(void)	Do nothing and return from spurious interrupts
defaultISR_2(void)	Turn off level and return from spurious interrupts
external_interrupt_init(function)	Given function to handle all spurious conditions
external_interrupt_enable(enable)	Enable the interrupt controller on the MPC555
IMB3_init(function)	Given function to handle all spurious LVL8..31 interrupts
LVL_is_empty(priority , function)	Check availability of an interrupt level.
IRQ Functionality Setup	
IRQ_init(function1 , function2)	Sets IRQ pins for either GPIO or IRQ functionality.
IRQ General Purpose Input / Output Functionality	
IRQ_port_direction_set(direction)	Set the direction of the IRQ port as either input or output.
IRQ_pin_direction_set(IRQ pin , direction)	Set the direction of a single IRQ pin as either input or output.
IRQ_port_direction_get(void)	Get the current direction of the IRQ port.
IRQ_pin_direction_get(IRQ pin)	Get the current direction of a single IRQ pin.
IRQ_port_set(data)	**SET VALUE OF THE IRQ PORT. WRITES TO INPUTS HAVE NO EFFECT.**
IRQ_pin_set (IRQ pin , pin_data)	Set value of a single IRQ pin. Writes to inputs have no effect.
IRQ_port_get(void)	Get value of the IRQ port. Outputs return current driven value.
IRQ_pin_get(IRQ pin)	Get value of a single IRQ pin. Outputs return current driven value.
IRQ as IRQ	
IRQ_isr_enable(priority, enable)	Enables/Disables a single IRQ level.
IRQ_isr_set(priority, function)	Sets a ISR for a given IRQ level.

* Functions in gray are a part of the ISR555 library and are not disributed with the SS555-DK. To purchase copies of the ISR555 please contact Intec Automation Inc.

MPIO	
MPIO_port_direction_set(direction)	Set the direction of the MPIO port as either input or output.
MPIO_pin_direction_set(MPIO pin , direction)	Set the direction of a single MPIO pin as either input or output.
MPIO_port_direction_get(void)	Get the current direction of the MPIO port.
MPIO_pin_direction_get(MPIO pin)	Get the current direction of a single MPIO pin.
MPIO_port_set(data)	Set value of the MPIO port. Writes to inputs have no effect.
MPIO_pin_set (MPIO pin , pin_data)	Set value of a single MPIO pin.Writes to inputs have no effect.
MPIO_port_get(void)	Get value of MPIO port. Outputs return current driven value.
MPIO_pin_get(MPIO pin)	Get value of an MPIO pin. Outputs return current driven value.
PIT / RTC combined functions	
PITRTclk_init(Clock Source, Division Factor)	sets combined PIT/RTC properties
PIT only functions	
PIT_get( )	returns time until next PIT interrupt in µs
PIT_tic_time( );	Returns the number of µs per PIT tic.
PIT_speed_set(speed);	Sets the resolution of the PIT.
PIT_period (tics);	Sets the number of PIT tics before the PIT timesout.
PIT_enable (enable)	Enables/disables the PIT timer
PIT_priority_set (priority, function)	Sets the interrupt level and the PIT interrupt subroutine.
PIT_isr_set(PIT, Error)	Set the PIT interrupt vector table
PIT_isr(void)	Runs the correct PIT isr and clears the interrupt flag
ReturnType PIT_interrupt_enable (enable);	Enables/disables the PIT interrupt.
PWM as PWM	
MIOS_init(clock_prescaler)	Sets the clock that is shared by the PWM and DA pins.
PWM_init(PWMpin , function)	Initialize a single PWM pin for either PWM or GPIO operation
PWM_period(PWMpin, clock_prescaler , period)	Sets the period of a single PWM pin.
PWM_pulse(PWMpin , pulse)	Sets the high time of a single PWM pin
PWM General Purpose Input / Output Functionality	
PWM_port_direction_set(direction)	Set the direction of the PWM port as either input or output.
PWM_pin_direction_set(PWM pin , direction)	Set the direction of a single PWM pin as either input or output.
PWM_port_direction_get(void)	Get the current direction of the PWM port.
PWM_pin_direction_get(PWM pin)	Get the current direction of a single PWM pin.
PWM_port_set(data)	Set value of the PWM port. Writes to inputs have no effect.
PWM_pin_set (PWM pin , pin_data)	Set value of a single PWM pin. Writes to inputs have no effect.
PWM_port_get(void)	Get value of PWM port. Outputs return current driven value.
PWM_pin_get(PWM pin)	Get value of a PWM pin. Outputs return current driven value.
QADC	
ADC_init(mode , port , pin);	Setup a single ADC port for either single or multiple scan.
adc_get(port, pin);	Get the value of a single read of 1 adc pin
adc_64scan_get(port);	Get the value of a 64 reads of 1 adc pin
adc_avg_get(port);	Get the average of 64 reads of 1 adc pin
RTC only functions	
RTC_get( );	Returns current value of the PIT
RTC_write(*RTC_value*);	Sets the current value, in seconds, of the RTC.
RTC_priority_set(*priority;function*)	Setup the interrupt for the RTC alarm.
RTC_isr_set(Alarm, OncePerSecond, Error)	Setup the RTC interrupt vector table
RTC_isr(void)	Runs the correct RTC isr and clears the interrupt flag
RTC_sec_int(*SEC_enable*);	Enables/disables the once per second interrupt.
RTC_alr_int(*enable; RTC_alarm*)	Setup the once RTC timer alarm interrupt.
SCI	
ser_int_en(boolean enable)	Enable SCI1 to interrupt the SS555
ser_rst_en(boolean enable)	Enable SCI1 to reset the SS555
ser_ack(boolean ack)	SS555 sends an acknowledge signal to the PC.
TB / DEC combined functions	

TMBCLK_set(TBS)	Sets source of TB/DEC clock
TMBCLK_en(TMB_enable)	Enables the TMB clock
TB only functions	
TB_set(tb)	sets the current value of the TB
TB_get( )	returns the current value of the TB
TB_priority_set(priority , function);	Setup of TB interrupt subroutine
TB_isr_set(TB_ref0 , TB_ref1, Error)	Setup the RTC interrupt vector table
TB_isr(void)	Runs the correct RTC isr and clears the interrupt flag
TB_ref_set(ref , tbref, enable)	Setup and enable TB reference registers to interrupt
Timing	
sysclk_set(freq_select);	Sets system frequency of the MPC555.

5. COL

Change Of Lock

General Description

The Change of Lock interrupt fires when the MPC555 clock oscillator loses lock. If the 4MHz external clock and internal Phase Lock Loop (PLL) become unstable, a COL interrupt occurs. For more information, see section 6.6 in the MPC555 User's Manual.

Pin Location

The COL is an internal module, having no external pins, although it is dependent on the XFC, XTAL, and EXTAL, the external clock oscillator pins.

Advanced Users

Functions

COL functions	
COL_priority_set(priority , function)	Set the level to interrupt
COL_isr_set(COL , Error)	Setup the COL interrupt vector table
COL_isr(void)	Execute correct COL isr
COL_interrupt_enable(enable)	Enable/disable the COL interrupt

1. COL_priority_set – COL Interrupt Priority Set

Description: Setup the interrupt for the change of lock alarm.

Syntax: ReturnType COL_priority_set(*UInt8 priority; IRQFuncType function*);

Prototype in: ..gcc\lib\intec-lib\ISRHandler.h

Parameters:

	priority	Priority level for the COL alarm to interrupt on. **Range**: 0..7
	function	Function to run when the COL interrupts.

Returns: Returns a success or failure code. See ReturnTypes,(page 2) for more information.

Example:

```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "ISRHandler.h"     //  RTC functions
                    :
external_interrupt_init( defaultISR_1 );      // Setup interrupt vector table
COL_isr_set( COL ; defaultISR_1 );            // Setup COL vector table
COL_priority_set( 5 , COL_isr );              // COL Interrupts on LVL5
                    :
```

Comments:

Location of Pins: The COL is an internal module, having no external pins.

Demo Programs: COL_isr.ipj

See Also: COL_isr_set; COL_isr; COL_interrupt_enable

☐ *RTL555* ☑ *ISR555* ☐ TBD

2. COL_isr_set – COL Interrupt Vector Table Set

Description: Sets the vector table with the functions to run when the COL interrupts.

Syntax: ReturnType COL_isr_set (*IRQFuncType COL , IRQFuncType Error*);

Prototype in: ..gcc\lib\intec-lib\ISRHandler.h

Parameters:

COL	Function to run when a COL interrupt occurs.
Error	Function to run when a spurious interrupt occurs.

Returns: Returns a success or failure code. See ReturnTypes (page 6) for more information.

Example:

```
#include "m_common.h"      // typedefs and some #define's for the 555
#include "timers.h"        //
void COL( void ){
                   :
}

main{
                   :
external_interrupt_init( defaultISR_1 );      // Setup interrupt vector table
COL_isr_set( COL ; defaultISR_1 );            // Setup COL vector table
COL_priority_set( 5 , COL_isr );              // COL Interrupts on LVL5
                   :
                   :
```

Comments: None.

Location of Pins: The COL is an internal module, having no external pins.

Demo Programs: COL_isr.ipj

See Also: COL_priority_set; COL_isr; COL_interrupt_enable

☐ *RTL555* ☑ *ISR555* ☐ TBD

3. COL_isr – COL Interrupt Subroutine

Description: Determines the source of the COL interrupt.

Syntax: void COL_isr (*void*);

Prototype in: ..gcc\lib\intec-lib\ISRHandler.h

Parameters: None.

Returns: None.

Example:

```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "timers.h"         //
void COL( void ){
                    :
}

main{
                    :
external_interrupt_init( defaultISR_1 );      // Setup interrupt vector table
COL_isr_set( COL; defaultISR_1 );             // Setup COL vector table
COL_priority_set( 5 , COL_isr );              // COL Interrupts on LVL5
                    :
```

Comments: None.

Location of Pins: The COL is an internal module, having no external pins.

Demo Programs: COL_isr.ipj

See Also: COL_priority_set; COL_isr_set COL_interrupt_enable

☐ *RTL555* ☑ *ISR555* ☐ TBD

4. COL_interrupt_enable

Description: Enables/disables the COL's ability to interrupt.

Syntax: ReturnType COL_interrupt_enable (*boolean enable*);

Prototype in: ..gcc\lib\intec-lib\ISRHandler.h

Parameters: **enable** COL interrupt enable
Range DISABLE/ENABLE
DISABLE = 0 ENABLE = 1

Returns: Returns a success or failure code. See ReturnTypes,(page 2) for more information.

Example:

```
#include "m_common.h"      // typedefs and some #define's for the 555
#include "timers.h"        //
void COL( void ){
                   :
}

main{
                   :
external_interrupt_init( defaultISR_1 );     // Setup interrupt vector table
COL_isr_set( COL; defaultISR_1 );            // Setup COL vector table
COL_priority_set( 5 , COL_isr );             // COL Interrupts on LVL5
COL_interrupt_enable( ENABLE );              // Enable the COL interrupt
                   :
```

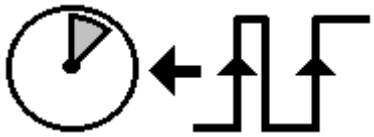
Comments:

Location of Pins: The COL is an internal module, having no external pins.

Demo Programs: COL_isr.ipj

See Also: COL_priority_set; COL_isr; COL_isr_set; COL_isr; COL_isr;

☐ *RTL555* ☑ *ISR555* ☐ TBD



6. DA

Double Action

General Description

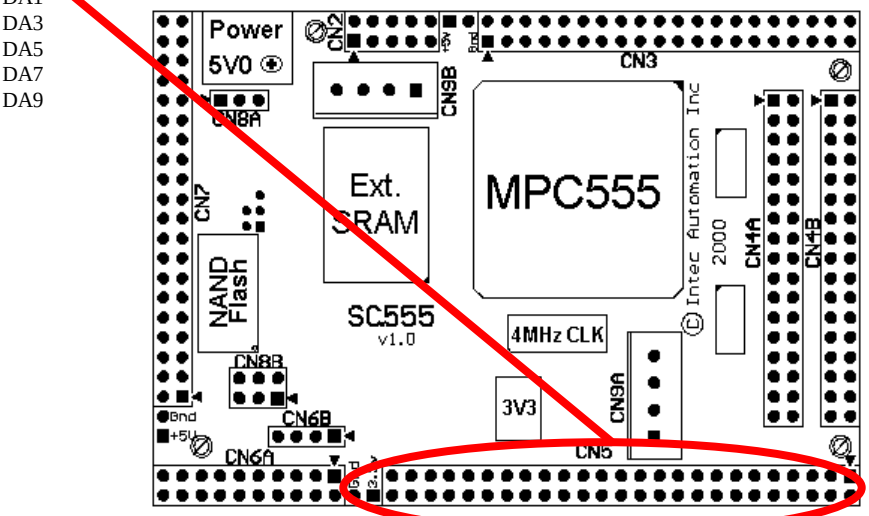
The double action submodule has six modes of operation. The double action pins can be configured in either disabled mode, pulse width or period measurement mode, input capture mode, or single or continuous output pulse mode.

Pin Location

CN5

DA0	1	2	DA1
DA2	3	4	DA3
DA4	5	6	DA5
DA6	7	8	DA7
DA8	9	10	DA9
	11	12	
	13	14	
	15	16	
	17	18	
	19	20	
	21	22	
	23	24	
	25	26	
	27	28	
	29	30	
	31	32	
	33	34	
	35	36	
	37	38	
	39	40	
	41	42	
	43	44	
	45	46	
	47	48	
	49	50	

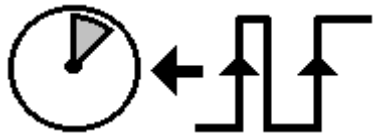
2 x 25



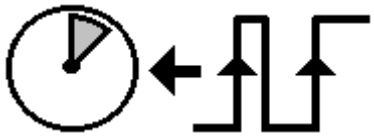
Advanced Users

Functions

DA functions	
DA_init(DApin, mode , counter , polarity)	Initialize a single Double Action pin
DA_write(DApin , data_reg , data)	Write to a register from one DA pin
DA_read(DApin, data_reg)	Read a register from one DA pin
MIOS Interrupt Functions	
MIOS_init(which_MIOS , function)	Set the spurious MIOS ISR.
MIOS_priority_set(which_MIOS,priority,function)	Set the level to interrupt
MIOS_isr_set(which_MIOS,which_pin,function, enable)	Setup and enable a single entry in the MIOS interrupt vector table



MIOS0_isr(void)	Execute correct MIOS0 isr
MIOS1_isr()	Execute correct MIOS1 isr



1. MIOS_interrupt_init –MIOS Interrupt Vector Table Init

Description: Fill in the given MIOS interrupt vector table with the default interrupt function specified the user. The function can be an ISR provided with the library or a user written function.

Syntax: ReturnType MIOS_init(*boolean which_MIOS , IRQFuncType function);*

Prototype in: ..gcc\lib\intec-lib\ISRHandler.h

Parameters:

which_MIOS Discriminates between MIOS0 and MIOS1.
Range: 0,1
0=MIOS0 1=MIOS1

function Function to run when a spurious interrupt occurs within the MIOSx submodule.

Returns: Returns a success or failure code. See ReturnTypes,(page 2) for more information.

Example:

```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "ISRHandler.h"     //  RTC functions
                    :
external_interrupt_init( defaultISR_2 );        // Fill vector table with function
   IMB3_init( defaultISR_2 , 32 );              // Use 32 interrupts and fill
                                                // 8,16,24,32 are valid numbers
                                                // of interrupt levels to enable
//Setup MIOS 0 Interrupts
MIOS_interrupt_init( 0 , defaultISR_1 );  // Set default MIOS0 table
MIOS_priority_set( 0 , 14 , MIOS0_isr );        // MIOS0 to interrupt on level
                                                // 14.  MIOS0_isr determines
                                                // which pin within the module
                                                // caused the interrupt.
MIOS_isr_set( 0 , CS0 , CS0_isr , ENABLE );   // MIOS0 CS0 interrupts are
                                                // enabled.  CS0_isr called
                                                // when CS0 interrupts.
                    :
```

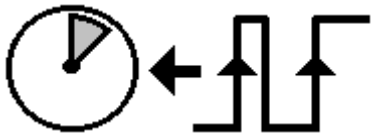
Comments:

Location of Pins: Schematics Sheet 7/8 CN5 Pins 0-9,13-22

Demo Programs:

See Also:

☐ *RTL555* ☑ *ISR555* ☐ TBD



2. MIOS_priority_set - MIOS Interrupt Priority Set

Description: Set the interrupt level (0-31) and the function to determine MIOS interrupt source of either the MIOS0 or MIOS1 submodule. . MIOSx_isr is provided as a function to determine the MIOS source.

Syntax: ReturnType MIOS_priority_set(*boolean which_MIOS , UInt8 priority , IRQFuncType function);*

Prototype in: ..gcc\lib\intec-lib\ISRHandler.h

Parameters:

which_MIOS Discriminates between MIOS0 and MIOS1.
Range: 0,1
0=MIOS0 1=MIOS1

priority Priority level for the COL alarm to interrupt on.
Range: 0..7

function Function to run when the COL interrupts.

Returns: Returns a success or failure code. See ReturnTypes,(page 2) for more information.

Example:

```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "ISRHandler.h"     //  RTC functions
                    :
external_interrupt_init( defaultISR_2 );        // Fill vector table with function
   IMB3_init( defaultISR_2 , 32 );              // Use 32 interrupts and fill
                                                // 8,16,24,32 are valid numbers
                                                // of interrupt levels to enable
//Setup MIOS 0 Interrupts
MIOS_interrupt_init( 0 , defaultISR_1 );        // Set MIOS0 spurious interrupt
MIOS_priority_set( 0 , 14 , MIOS0_isr );        // MIOS0 to interrupt on level
                                                // 14.  MIOS0_isr determines
                                                // which pin within the module
                                                // caused the interrupt.
MIOS_isr_set( 0 , CS0 , CS0_isr , ENABLE );     // MIOS0 CS0 interrupts are
                    :
```

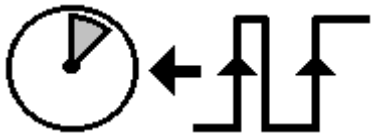
Comments:

Location of Pins: Schematics Sheet 7/8 CN5 Pins 0-9,13-22

Demo Programs:

See Also:

☐ *RTL555* ☑ *ISR555* ☐ TBD



3. MIOS_isr_set –MIOS Interrupt Vector Table Set

Description: Sets the vector table with the function to run when the MIOSx interrupts. The pins are all #defined and it is strongly recommended that these are used to avoid errors in the setup of the MIOS vector table. Because of the large number of MIOS interrupt sources that are all routed through the same interrupt level, the vector table is setup one field in the array at a time. This means that for every MIOSx source that interrupts, this function must be called once. This function also enables and disables the individual interrupting pins. The correct function must again be specified if the interrupt is to be re-enabled.

Syntax: ReturnType MIOS_isr_set (*boolean which_MIOS , UInt16 WhichPin , IRQFuncType function , boolean enable*);

Prototype in: ..gcc\lib\intec-lib\ISRHandler.h

Parameters:

which_MIOS	Discriminates between MIOS0 and MIOS1. Range: 0,1 0=MIOS0 1=MIOS1
WhichPin	Discriminates between the MIOSx sources
function	ISR to run when the MIOSx pin is the source of the interrupt.
enable	Enable or disable the ability of the pin to interrupt.

Returns: Returns a success or failure code. See ReturnTypes (page 6) for more information.

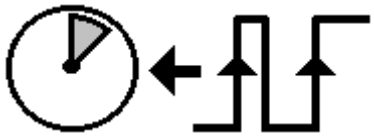
Example:

```
#include "m_common.h"      // typedefs and some #define's for the 555
#include "ISRHandler.h"    //  RTC functions
                  :
//Setup MIOS 0 Interrupts
MIOS_interrupt_init( 0 , defaultISR_1 );     // Set MIOS0 spurious interrupt
MIOS_priority_set( 0 , 14 , MIOS0_isr );     // MIOS0 to interrupt on level
                                             // 14.  MIOS0_isr determines
                                             // which pin within the module
                                             // caused the interrupt.
MIOS_isr_set( 0 , CS0 , CS0_isr , ENABLE );  // MIOS0 CS0 to interrupt
                  :
```

Comments:

Location of Pins: Schematics Sheet 7/8 CN5 Pins 0-9,13-22
Demo Programs:
See Also:

☐ *RTL555* ☑ *ISR555* ☐ TBD



4. MIOS0_isr – MIOS 0 Interrupt Subroutine

Description: Determines the source of the MIOS0 interrupt.

Syntax: void MIOS0_isr (*void*);

Prototype in: ..gcc\lib\intec-lib\ISRHandler.h

Parameters: None.

Returns: None.

Example:

```
#include "m_common.h"      // typedefs and some #define's for the 555
#include "timers.h"        //
void PWM0_isr( void ){
                   :
}
                   :
//Setup MIOS 0 Interrupts
MIOS_interrupt_init( 0 , defaultISR_1 );      // Set MIOS0 spurious interrupt
MIOS_priority_set( 0 , 14 , MIOS0_isr );  // MIOS0 to interrupt on level
                                          // 14.  MIOS0_isr determines
                                          // which pin within the module
                                          // caused the interrupt.
MIOS_isr_set( 0 , PWM0 , PWM0_isr , ENABLE ); // MIOS0 PWM0 interrupt
                   :
```

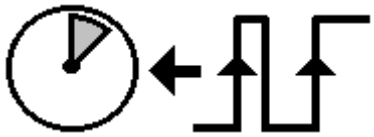
Comments:

Location of Pins: Schematics Sheet 7/8 CN5 Pins 0-9,13-22

Demo Programs:

See Also:

☐ *RTL555* ☑ *ISR555* ☐ TBD



5. MIOS1_isr – MIOS 1 Interrupt Subroutine

Description: Determines the source of the MIOS1 interrupt.

Syntax: void MIOS1_isr (*void*);

Prototype in: ..gcc\lib\intec-lib\ISRHandler.h

Parameters: None.

Returns: None.

Example:

```
#include "m_common.h"      // typedefs and some #define's for the 555
#include "timers.h"        //
void PWM4_isr( void ){
                   :
}
                   :
//Setup MIOS 0 Interrupts
MIOS_interrupt_init( 0 , defaultISR_1 );      // Set MIOS0 spurious interrupt
MIOS_priority_set( 1 , 31 , MIOS1_isr );  // MIOS1 to interrupt on level
                                          // 31.  MIOS1_isr determines
                                          // which pin within the module
                                          // caused the interrupt.
MIOS_isr_set( 1 , PWM4 , PWM4_isr , ENABLE ); // MIOS1 PWM4 interrupt
                   :
```

Comments:

Location of Pins: Schematics Sheet 7/8 CN5 Pins 0-9,13-22

Demo Programs:

See Also:

☐ *RTL555* ☑ *ISR555* ☐ TBD

7. DEC

Decrementer

General Description

The DEC generates a non-maskable interrupt when it decrements to zero. The DEC can be set to generate interrupts from once every microsecond to approximately once every 13 hours. The DEC is not affected by HRESET, and therefore will hold its value after a debugger initialized reset and will continue to operate in low power modes. For more information, see section 6.6 in the MPC555 User's Manual.

Pin Location

The DEC is an internal module, having no external pins.

Advanced Users

Using F_{sys} as the clock source to the TMBCLK is more difficult than using the crystal oscillator. With the oscillator, the desired time of the interrupt is entered in terms of microseconds. With the f_{sys} the user has to enter the desired time in terms of clock tics. Changing the f_{sys} would necessitate resetting the TMBCLK. The following example assumes that the crystal oscillator is driving the MPC555. Using the backup clock will be discussed later.

$$f_{sys} = \frac{4MHz \cdot MF}{DFNH \text{ or DFNL}}$$

The values of these parameters are kept in a hidden structure called SC555.
SC555.MF = current multiplication factor
SC555.DFNHL = current division factor being used (value of DFNH or DFNL)

$$f_{TMBCLK} = \frac{f_{sys}}{16} \qquad 1\,\text{clock tic} = \frac{16}{f_{sys}} \qquad REFx \text{ value} = \frac{\text{desired time}}{1\,\text{clock tic}}$$

Example for MF=10
DFNH = 1
1 second desired

$$1\,\text{clock tic} = \frac{16*1}{4\text{MHz}*10} = 0.4\mu s \qquad REF0 \text{ value} = \frac{1s}{0.4us} = 2500000$$

Example for MF=4
DFNL = 256
1 second desired

$$1\,\text{clock tic} = \frac{16*256}{4\text{MHz}*4} = 256\mu s \qquad REF0 \text{ value} = \frac{1s}{256us} \approx 3906$$

Functions

TB / DEC combined functions	
TMBCLK_set(Clock Source , ClockSourceEnable)	sets combined TB/DEC properties
DEC only functions	
DEC_set(time)	sets the current value of the DEC in µs
DEC_get( )	returns the current value of the DEC

1. TMBCLK_set –Timebase Clock Set

Description: Set the current TMB clock source and enables or disables the clock. This clock is shared by both the Time Base and Decrementer timers.

Syntax: ReturnType TMBCLK_set (*boolean TBS, boolean TMB_enable*);

Prototype in: ..gcc\lib\intec-lib\timers.h

Parameters:

TBS	Timebase/Decrementer Clock source. Should usually be run from the OSCM (the main crystal oscillator). Range = 0 = OSCM(4MHz) 1=F_{SYS} (system clock)
TMB_enable	Timebase/Decrementer Clock enable Range DISABLE/ENABLE DISABLE = 0 ENABLE = 1

Returns: Returns a success or failure code. See ReturnTypes (page 6) for more information.

Example:

```
#include "m_common.h"      // typedefs and some #define's for the 555
#include "timers.h"        //
                   :
TMBCLK_set( OSCM , ENABLE);
                              // Time base clock source=oscillator (4MHz)
DEC_set( 0x1000 );         //  Set the Decrementer to a value of 0x1000
                   :
```

Comments: None.

Location of Pins: The DEC is an internal module, having no external pins.

Demo Programs: DEC.ipj

See Also: DEC_get; DEC_set; TB_set; TB_get; TB_ref_set; TB_priority_set

---

☑ *RTL555* ☐ *ISR555* ☐ TBD

2. DEC_set – Decrementer Set

Description: Set the current value of the time base register.

Syntax: ReturnType DEC_set (*UInt64 period*);

Prototype in: ..gcc\lib\intec-lib\timers.h

Parameters: **period** New time to setup the decrementer.
Due to constraints on variable size, this is equal to time-1.
Range = 0 – 0xFFFF FFFF FFFF FFFF

Returns: Returns a success or failure code. See ReturnTypes (page 6) for more information.

Example:

```
#include "m_common.h"      // typedefs and some #define's for the 555
#include "timers.h"        //
                   :
TMBCLK_set( OSCM );  // Time base clock source=oscillator (4MHz)
DEC_set( 0x1000 );   //  Set the Decrementer to a value of 0x1000
                   :
```

Comments: None.

Location of Pins: The DEC is an internal module, having no external pins.

Demo Programs: DEC.ipj

See Also: DEC_get;

☑ *RTL555* ☐ *ISR555* ☐ TBD

3. DEC_get – Decrementer get

Description: Gets the current value of the time base register.

Syntax: UInt64 DEC_get (void);

Prototype in: ..gcc\lib\intec-lib\timers.h

Parameters: None.

Returns: Current value of the decrementer timer.

Example:

```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "timers.h"         //
                    :
UInt64 DEC_time;
TMBCLK_set( OSCM );    // Time base clock source=oscillator (4MHz)
DEC_set( 0x1000 );     //  Set the Decrementer to a value of 0x1000
                    :
DEC_time=DEC_get();    // Get the current DEC value
                    :
```

Comments: None.

Location of Pins: The DEC is an internal module, having no external pins.

Demo Programs: DEC.ipj

See Also: DEC_set;

☑ *RTL555* ☐ *ISR555* ☐ TBD

4. DEC_isr_set – Decrementer Interrupt Subroutine Set

Description: Sets the user written subroutine to run when the decrementer interrupts.

Syntax: RturnType DEC_isr_set (*IRQFuncType function*);

Prototype in: ..gcc\lib\intec-lib\timers.h

Parameters: IRQFuncType Function to run when a DEC interrupt occurs.

Returns: Returns a success or failure code. See ReturnTypes (page 6) for more information.

Example:

```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "timers.h"         //
                    :
IRQFuncType DEC_isr( void ){
                    :
}
                    :
UInt64 DEC_time;
TMBCLK_set( OSCM );    // Time base clock source=oscillator (4MHz)
DEC_set( 0x1000 );     //  Set the Decrementer to a value of 0x1000
DEC_isr_set(DEC_isr);
                    :
```

Comments: None.

Location of Pins: The DEC is an internal module, having no external pins.

Demo Programs: DEC.ipj

See Also: DEC_set; DEC_get

☐ *RTL555* ☑ *ISR555* ☐ TBD

---

8. Flash Programming

General Description

On the MPC555 there are two signals that must be properly conditioned before writes to the Flash EPROM are accepted; EPEE and VPP. There is one function in the runtime library that enables/disables writes to the flash programming. This function is not needed to load programs into flash using either PROGPPC or VisionCLICK. See the section Getting Started with PROGPPC in this manual for more details.

Pin Location

There are no external pins associated with the Flash EPROM although there is an LED indicator on the board that lights up when flash programming is enabled.

Functions

Flash Programming	
ReturnType flash_enable(enable);	Enables/disables flash programming on the SS555. Sets correct levels on both VPP and EPEE.

1. flash_enable

Description: Enable/Disable the Flash EPROM programming on the MPC555

Syntax: ReturnType flash_enable (*boolean enable*);

Prototype in: ..gcc\lib\intec-lib\cpld.h

Parameters:

enable	Enables/Disables writes to Flash EPROM. Range = ENABLE, DISABLE DISABLE = 0 ENABLE = 1

Returns: Returns a success or failure code. See Section x.x. - Return Types for more information.

Example:

```
#include "m_common.h"          // typedefs and some #define's for the 555
#include "cpld.h"              // ser_int_en();ser_rst_en;ser_ack()
                :
flash_enable( ENABLE );        // Enable Flash programming on the 555
flash_enable ( DISABLE );      // Disable Flash programming on the 555
                :
```

Comments: None.

Location of Pins: None.

Demo Programs: flash.ipj

See Also:

☑ *RTL555* ☐ *ISR555* ☐ TBD

9. Interrupt Controller

General Description

An interrupt controller handles the interrupts from the internal modules of the MPC555. For detailed information, see the separate interrupt documentation.

Pin Location

The interrupt controller is an internal modul, containing no pins.

Functions

IRQ Functionality	
IRQ_init(function1 , function2)	
General Purpose Input / Output Functionality	
Port	**Pin**
Initialization of the IRQ GPIO	
IRQ_port_direction_set(direction)	ReturnType IRQ_pin_direction_set(UInt8 IRQ pin , boolean direction)
IRQ_port_direction_get(void)	boolean IRQ_pin_direction_get(UInt8 IRQ pin)
IRQ as an Output from the MPC555	
IRQ_port_set(data)	ReturnType IRQ_pin_set (UInt8 IRQ pin , boolean pin_data)
IRQ as an Input to the MPC555	
IRQ_port_get(void)	boolean IRQ_pin_get(UInt8 IRQ pin)
IRQ as IRQ	
IRQ_isr_enable(priority, enable)	ReturnType IRQ_isr_set(UInt8 priority, IRQFuncType function)

1. defaultISR_1 - Default ISR 1

Description: This function specifies how the interrupt controller recovers form a spurious interrupt. defaultISR_1 simply returns from the interrupt.

Syntax: void defaultISR_1 (*void*);

Prototype in: **…**gcc\lib\intec-lib\IRQ.h

Parameters: None.

Returns: None.

Example:

```
#include "m_common.h"              // typedefs and some #define's for the 555
#include "IRQ.h"                   // IRQ_init(), IRQ_port_direction_set()..
                    :
external_interrupt_init( defaultISR_1 );    // Return from spurrious ISRs
                    :
```

Comments:

Location of Pins: None.

Demo Programs: IRQ_isr.ipj; IRQ_isr_all.ipj; blinkall.ipj

See Also: defaultISR_2; external_interrupt_init; external_interrupt_enable; IMB3_init; LVL_is_empty

---

☑ *RTL555* ☐ *ISR555* ☐ TBD

2. defaultISR_2 - Default ISR 2

Description: This function specifies how the interrupt controller recovers form a spurious interrupt. defaultISR_2 turns off the interrupt level that caused the spurious interrupt and returns. This ensures that the same spurious interrupt will not happen again.

Syntax: void defaultISR_2 (*void*);

Prototype in: **…**gcc\lib\intec-lib\IRQ.h

Parameters: None.

Returns: None.

Example:

```
#include "m_common.h"              // typedefs and some #define's for the 555
#include "IRQ.h"                   // IRQ_init(), IRQ_port_direction_set()..
                    :
external_interrupt_init( defaultISR_1 );    // Return from spurrious ISRs
                    :
```

Comments: Please note that if one of the IMB3 interrupts misfires and defaultISR_2 was used during the IMB3_init function, all interrupts from LVL8-31 are disabled. The only way to re-enable is to run the IMB3_init function again.

Location of Pins: None.

Demo Programs: IRQ_isr.ipj; IRQ_isr_all.ipj; blinkall.ipj

See Also: defaultISR_1; external_interrupt_init; external_interrupt_enable; IMB3_init; LVL_is_empty

---

☑ *RTL555* ☐ *ISR555* ☐ TBD

3. external_interrupt_init - Interrupt Vector Table Init

Description: Fills in the vector table that handles all external interrupts with the function specified by the user. These functions are run when a spurious interrupt occurs. There are a few default ISRs provided with the library, or the user may specify their own function.

Syntax: ReturnType external_interrupt_init(*IRQFuncType function*);

Prototype in: **…**gcc\lib\intec-lib\IRQ.h

Parameters: **function** Function to filll the interrupt vector table
Must be of type **void** func **(void).**

Returns: Returns a success or failure code. See ReturnTypes,(page 6) for more information.

Example:

```
#include "m_common.h"          // typedefs and some #define's for the 555
#include "IRQ.h"               // IRQ_init(), IRQ_port_direction_set()..
                :
external_interrupt_init( defaultISR_1 );    // Return from spurrious ISRs
                :
```

Comments:

Location of Pins: None.

Demo Programs: IRQ_isr.ipj; IRQ_isr_all.ipj; blinkall.ipj

See Also: defaultISR_1; defaultISR_2; external_interrupt_enable; IMB3_init; LVL_is_empty

---

☑ *RTL555* ☐ *ISR555* ☐ TBD

4. external_interrupt_enable - External Interrupt Enable

Description: Enable/Disable the interrupt controller.

Syntax: ReturnType external_interrupt_init(*boolean enable*);

Prototype in: **…**gcc\lib\intec-lib\IRQ.h

Parameters: **enable** Enable/Disable the interrupt controller.
Range DISABLE/ENABLE
DISABLE = 0 ENABLE = 1

Returns: Returns a success or failure code. See ReturnTypes,(page 6) for more information.

Example:

```
#include "m_common.h"          // typedefs and some #define's for the 555
#include "IRQ.h"               // IRQ_init(), IRQ_port_direction_set()..
                :
external_interrupt_init( defaultISR_1 );   // Return from spurrious ISRs
external_interrupt_enable( ENABLE );       // Enable the interrupt controller

                :
```

Comments:

Location of Pins: None.

Demo Programs: IRQ_isr.ipj; IRQ_isr_all.ipj; blinkall.ipj

See Also: defaultISR_1; defaultISR_2; external_interrupt_init; IMB3_init; LVL_is_empty

---

☑ *RTL555* ☐ *ISR555* ☐ TBD

5. IMB3_init - IMB3 Vector Table Initialize

Description: Fills in the vector table that handles all external interrupts on the IMB3 bus (LVL7..31) with the function specified by the user. These functions are run when a spurious interrupt occurs. There are a few default ISRs provided with the library, or the user may specify their own function. The IMB3 can not be used by all modules.

Syntax: ReturnType IMB3_init(*IRQFuncType function , UInt8 number_of_sources*);

Prototype in: **…**gcc\lib\intec-lib\IRQ.h

Parameters:

function Function to filll the IMB3 interrupt vector table
Must be of type **void** func **(void).**

number_of_sources Number of IMB3 sources to allow.
Range:8,16,24,32
8=LVL0-7 16=LVL0-15
24=LVL0-23 32=LVL0-31

Returns: Returns a success or failure code. See ReturnTypes,(page 6) for more information.

Example:

```
#include "m_common.h"            // typedefs and some #define's for the 555
#include "IRQ.h"                 // IRQ_init(), IRQ_port_direction_set()..
                :
external_interrupt_init( defaultISR_1 );      // Return from spurrious ISRs
external_interrupt_init( defaultISR_1,32);// Return from spurrious ISRs
                                          // 32 IMB3 levels used
                :
```

Comments: As more sources are added, the interrupt controller slows down. Use as few sources as possible.

Location of Pins: None.

Demo Programs: IRQ_isr.ipj; IRQ_isr_all.ipj; blinkall.ipj

See Also: defaultISR_1; defaultISR_2; external_interrupt_init; external_interrupt_enable; LVL_is_empty

---

☐ *RTL555* ☑ *ISR555* ☐ TBD

6. LVL_is_empty - Level Is Empty

Description: Compares the function located in a given level to the function specified by the user. A comparison is made between with the function that was specified with the external_interrupt_init(function) or IMB3_init(function, …) and LVL_is_empty(function).

Syntax: boolean LVL_is_empty(*UInt8 which_level*, *IRQFuncType function*);

Prototype in: **…**gcc\lib\intec-lib\ISRhandler.h

Parameters:

which_level	Level to test. Range: 0 - 31
function	Default ISR to test for.

Returns: Returns a 1 if the level still contains the default interrupt (empty), and will return a 0 if the function is different (full).

Example:

```
#include "m_common.h"              // typedefs and some #define's for the 555
#include "IRQ.h"                   // IRQ_init(), IRQ_port_direction_set()..
                    :
is_empty=0;
external_interrupt_init( defaultISR_1 );      // Return from spurrious ISRs
external_interrupt_enable( ENABLE );          // Enable the interrupt controller
is_empty = LVL_is_empty( 4 );                 // Ensure that level 4 is empty
if ( is_empty ){
                    :
```

Comments:

Location of Pins: None.

Demo Programs:

See Also: defaultISR_1; defaultISR_2; external_interrupt_init; external_interrupt_enable; IMB3_init; LVL_is_empty

---

☐ *RTL555* ☑ *ISR555* ☐ TBD

---

10. IRQ

Interrupt Request Pins

General Description

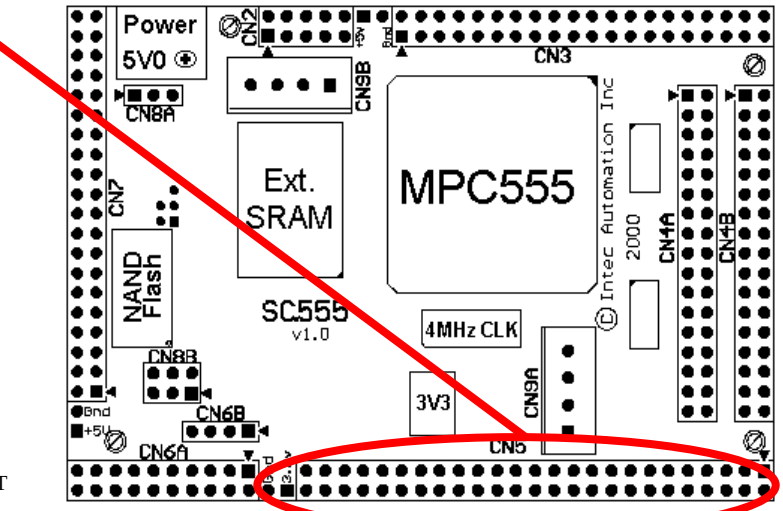
10 pin – 3 Volt interrupt request port. Can be set to either request servicing from the ISR or be used as 5V GPIO port. Reads and writes to the GPIO port are simultaneous. For more information see section 6.4 Interrupt Controller in the MPC555 user's manual.

Pin Location

CN5

	1	2	
	3	4	
	5	6	
	7	8	
	9	10	
	11	12	
	13	14	
	15	16	
	17	18	
	19	20	
	21	22	
	23	24	
	25	26	
	27	28	
	29	30	
	31	32	
	33	34	
	35	36	
	37	38	
FRZ	39	40	*IRQOUT
*IRQ0	41	42	*IRQ0
*IRQ2	43	44	*IRQ3
*IRQ4	45	46	*IRQ5
*IRQ6	47	48	*IRQ7
3V3	49	50	GND

2 x 25



Functions

IRQ Functionality	
IRQ_init(function1 , function2)	
General Purpose Input / Output Functionality	
Port	**Pin**
Initialization of the IRQ GPIO	
IRQ_port_direction_set(direction)	ReturnType IRQ_pin_direction_set(UInt8 IRQ pin , boolean direction)
IRQ_port_direction_get(void)	boolean IRQ_pin_direction_get(UInt8 IRQ pin)
IRQ as an Output from the MPC555	
IRQ_port_set(data)	ReturnType IRQ_pin_set (UInt8 IRQ pin , boolean pin_data)
IRQ as an Input to the MPC555	
IRQ_port_get(void)	boolean IRQ_pin_get(UInt8 IRQ pin)
IRQ as IRQ	
IRQ_isr_enable(priority, enable)	ReturnType IRQ_isr_set(UInt8 priority, IRQFuncType function)

1. IRQ_init

Description: Sets the IRQ port for either 3V3 IRQ operation or 5V0 GPIO operation. IRQ pins can only be set in two distinct blocks *IRQ0-*IRQ5 and *IRQOUT – FRZ.

Syntax: ReturnType IRQ_init(*UInt8 function1 , UInt8 function2*);

Prototype in: **…**gcc\lib\intec-lib\irq.h

Parameters:

function1 Sets the operation for pins 41-46(*IRQ/GPIO0-*IRQ/GPIO5)
Range = NA , GPIO , IRQ
0 = NA , 1 = GPIO, 2 = IRQ

function2 Sets the operation for pins 39-40 (*IRQOUT – FRZ)
Range = NA , GPIO , IRQ
0 = NA , 1 = GPIO, 2 = IRQ

Returns: Returns a success or failure code. See ReturnTypes,(page 6) for more information.

Example:

```
#include "m_common.h"          // typedefs and some #define's for the 555
#include "irq.h"               // IRQ_init(), IRQ_port_direction_set()..
                :
IRQ_init( GPIO, GPIO );        // Set all IRQ pins for GPIO operation
                :
```

Comments: GPIO pins are not continuous. Be careful that you do not mistake *IRQ6 for GPIO6 and *IRQ7 for GPIO7:

FRZ	39	40	*IRQOUT	GPIO6	39	40	GPIO7
*IRQ0	41	42	*IRQ0	GPIO0	41	42	GPIO1
*IRQ2	43	44	*IRQ3	GPIO2	43	44	GPIO3
*IRQ4	45	46	*IRQ5	GPIO4	45	46	GPIO5
*IRQ6	47	48	*IRQ7	*IRQ6	47	48	*IRQ7

Location of Pins: Schematics Sheet 7/8 CN5 Pins 39-50

Demo Programs: IRQ_gpio.ipj; IRQ_isr.ipj; IRQ_isr_all.ipj; blinkall.ipj

See Also: IRQ_port_direction_set; IRQ_port_direction_get; IRQ_port_set; IRQ_port_get; IRQ_pin_direction_set; IRQ_port_direction_get; IRQ_pin_direction_get; IRQ_pin_set; IRQ_pin_get; IRQ_isr_enable; IRQ_isr_set.

☑ *RTL555* ☐ *ISR555* ☐ TBD

2. IRQ_port_direction_set

Description: Sets the entire 8-bit port for I/O operation in one function call.

Syntax: ReturnType IRQ_port_direction_set(*UInt8 direction*);

Prototype in: ..gcc\lib\intec-lib\irq.h

Parameters: **direction** 8 bit hexadecimal number corresponding to the pattern of I/O operation for the entire port.
Range = 0x00– 0xFF
0=Input 1=Output

Returns: Returns a success or failure code. See ReturnTypes,(page 6) for more information.

Example:

```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "irq.h"            // IRQ_init(), IRQ_port_direction_set()..
                                    :
IRQ_init( GPIO, GPIO );   // Set all IRQ pins for GPIO operation
IRQ_port_direction_set( 0xFF );  // Set all IRQ pins for output operation
IRQ_port_direction_set( 0x00 );  // Set all IRQ pins for input operation
IRQ_port_direction_set( 0x55 );  // Set alternating IRQ pins as in/out
                                    :
```

Comments: GPIO pins are not continuous. Be careful that you do not mistake *IRQ6 for GPIO6 and *IRQ7 for GPIO7:

FRZ	39	40	*IRQOUT	GPIO6	39	40	GPIO7
*IRQ0	41	42	*IRQ0	GPIO0	41	42	GPIO1
*IRQ2	43	44	*IRQ3	GPIO2	43	44	GPIO3
*IRQ4	45	46	*IRQ5	GPIO4	45	46	GPIO5
*IRQ6	47	48	*IRQ7	*IRQ6	47	48	*IRQ7

Location of Pins: Schematics Sheet 7/8 CN5 Pins 39-50

Demo Programs: IRQ_gpio.ipj; IRQ_isr.ipj; IRQ_isr_all.ipj; blinkall.ipj

See Also: IRQ_init; IRQ_port_direction_get; IRQ_port_set; IRQ_port_get; IRQ_pin_direction_set; IRQ_port_direction_get; IRQ_pin_direction_get; IRQ_pin_set; IRQ_pin_get; IRQ_isr_enable; IRQ_isr_set.

☑ *RTL555* ☐ *ISR555* ☐ TBD

3. IRQ_pin_direction_set

Description: Sets an individual pin to either input or output operation.

Syntax: ReturnType IRQ_pin_direction_set(*UInt8 PIOpin,boolean direction*);

Prototype in: ..gcc\lib\intec-lib\irq.h

Parameters:

PIOpin Desired pin to set the direction.
Range = 0 – 7
Pin 0 = 0, Pin 1 = 1, Pin 2 = 2, …, Pin 6 = 6, Pin 7 = 7

direction Direction to set the given pin.
Range = INPUT or OUTPUT
0 = Input 1 = Output

Returns: Returns a success or failure code. See ReturnTypes,(page 2) for more information.

Example:

```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "irq.h"            // IRQ_init(), IRQ_port_direction_set()..
                    :
int pin;
IRQ_init( GPIO, GPIO );   // Set all IRQ pins for GPIO operation
IRQ_pin_direction_set( 7 , INPUT );  // Set GPIO pin 7 as an input
IRQ_pin_direction_set( 0 , OUPUT); // Set GPIO pin 0 as an outuput
                    :
```

Comments: GPIO pins are not continuous. Be careful that you do not mistake *IRQ6 for GPIO6 and *IRQ7 for GPIO7:

FRZ	39	40	*IRQOUT	GPIO6	39	40	GPIO7
*IRQ0	41	42	*IRQ0	GPIO0	41	42	GPIO1
*IRQ2	43	44	*IRQ3	GPIO2	43	44	GPIO3
*IRQ4	45	46	*IRQ5	GPIO4	45	46	GPIO5
*IRQ6	47	48	*IRQ7	*IRQ6	47	48	*IRQ7

Location of Pins: Schematics Sheet 7/8 CN5 Pins 39-50

Demo Programs: IRQ_gpio.ipj; IRQ_isr.ipj; IRQ_isr_all.ipj; blinkall.ipj

See Also: IRQ_init;IRQ_port_direction_set; IRQ_port_direction_get; IRQ_port_set; IRQ_port_get; IRQ_port_direction_get; IRQ_pin_direction_get; IRQ_pin_set; IRQ_pin_get; IRQ_isr_enable; IRQ_isr_set.

☑ *RTL555* ☐ *ISR555* ☐ TBD

4. IRQ_port_direction_get

Description: Gets the current setup of the entire 8-bit IRQ port in one function call.

Syntax: UInt8 IRQ_port_direction_get( );

Prototype in: ..gcc\lib\intec-lib\irq.h

Parameters: None

Returns: Hexadecimal number corresponding to the pattern of I/O operation for the entire port.
Range = 0x00– 0xFF
0=Input 1=Output

Example:

```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "irq.h"            // IRQ_init(), IRQ_port_direction_set()..
                    :
int direction;
IRQ_init( GPIO, GPIO );   // Set all IRQ pins for GPIO operation
IRQ_port_direction_set( 0xAA );// Set alternating IRQ pins for input/output
                    :
direction = IRQ_port_direction_get( );
                    :
```

Comments: GPIO pins are not continuous. Be careful that you do not mistake *IRQ6 for GPIO6 and *IRQ7 for GPIO7:

FRZ	39	40	*IRQOUT	GPIO6	39	40	GPIO7
*IRQ0	41	42	*IRQ0	GPIO0	41	42	GPIO1
*IRQ2	43	44	*IRQ3	GPIO2	43	44	GPIO3
*IRQ4	45	46	*IRQ5	GPIO4	45	46	GPIO5
*IRQ6	47	48	*IRQ7	*IRQ6	47	48	*IRQ7

Location of Pins: Schematics Sheet 7/8 CN5 Pins 39-50

Demo Programs: IRQ_gpio.ipj; IRQ_isr.ipj; IRQ_isr_all.ipj; blinkall.ipj

See Also: IRQ_init;IRQ_port_direction_set; IRQ_port_direction_get; IRQ_port_set; IRQ_port_get; IRQ_pin_direction_set; IRQ_pin_direction_get; IRQ_pin_set; IRQ_pin_get; IRQ_isr_enable; IRQ_isr_set.

☑ *RTL555* ☐ *ISR555* ☐ TBD

5. IRQ_pin_direction_get

Description: Gets the current setup of a single IRQ pin.

Syntax: boolean IRQ_pin_direction_get(*UInt8 PIOpin*)**;**

Prototype in: ..gcc\lib\intec-lib\irq.h

Parameters: **IRQpin** Desired pin to get.
Range = 0 – 7
Pin 0 = 0, Pin 1 = 1, Pin 2 = 2, …, Pin 6 = 6, Pin 7 = 7

Returns: Returns the current direction of the given pin.
Range = INPUT or OUTPUT
0=Input 1=Output

Example:
```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "irq.h"            // IRQ_init(), IRQ_port_direction_set()..
                    :
boolean direction;
IRQ_init( GPIO, GPIO );   // Set all IRQ pins for GPIO operation
IRQ_port_direction_set( 0xAA );// Set alternating IRQ pins for input/output
                    :
direction = IRQ_pint_direction_get( 7 ); // Get the direction of pin 7
                    :
```

Comments: GPIO pins are not continuous. Be careful that you do not mistake *IRQ6 for GPIO6 and *IRQ7 for GPIO7:

FRZ	39	40	*IRQOUT	GPIO6	39	40	GPIO7
*IRQ0	41	42	*IRQ0	GPIO0	41	42	GPIO1
*IRQ2	43	44	*IRQ3	GPIO2	43	44	GPIO3
*IRQ4	45	46	*IRQ5	GPIO4	45	46	GPIO5
*IRQ6	47	48	*IRQ7	*IRQ6	47	48	*IRQ7

Location of Pins: Schematics Sheet 7/8 CN5 Pins 39-50

Demo Programs: IRQ_gpio.ipj; IRQ_isr.ipj; IRQ_isr_all.ipj; blinkall.ipj

See Also: IRQ_init;IRQ_port_direction_set; IRQ_port_direction_get; IRQ_port_set; IRQ_port_get; IRQ_pin_direction_set; IRQ_port_direction_get; IRQ_pin_set; IRQ_pin_get; IRQ_isr_enable; IRQ_isr_set.

☑ *RTL555* ☐ *ISR555* ☐ TBD

6. IRQ_port_set

Description: Sets the data on all IRQ output pins in one command.

Syntax: ReturnType set_IRQ_port(*UInt8 data*);

Prototype in: ..gcc\lib\intec-lib\irq.h

Parameters: **data** Output data to write to the IRQ port.
Range = 0x00 to 0xFF
0=GND=low 1=5V0=high

Returns: Returns a success or failure code. See ReturnTypes,(page 2) for more information.

Example:
```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "irq.h"            // IRQ_init(), IRQ_port_direction_set()..
                    :
IRQ_init( GPIO, GPIO );   // Set all IRQ pins for GPIO operation
IRQ_port_direction_set( 0xFF ); // Set all IRQ pins for output operation
                    :
IRQ_port_set( 0xAA );  // Alternate the IRQ pins HI/LO
                    :
```

Comments: GPIO pins are not continuous. Be careful that you do not mistake *IRQ6 for GPIO6 and *IRQ7 for GPIO7:

FRZ	39	40	*IRQOUT	GPIO6	39	40	GPIO7
*IRQ0	41	42	*IRQ0	GPIO0	41	42	GPIO1
*IRQ2	43	44	*IRQ3	GPIO2	43	44	GPIO3
*IRQ4	45	46	*IRQ5	GPIO4	45	46	GPIO5
*IRQ6	47	48	*IRQ7	*IRQ6	47	48	*IRQ7

Location of Pins: Schematics Sheet 7/8 CN5 Pins 39-50

Demo Programs: IRQ_gpio.ipj; IRQ_isr.ipj; IRQ_isr_all.ipj; blinkall.ipj

See Also: IRQ_init;IRQ_port_direction_set; IRQ_port_direction_get; IRQ_port_get; IRQ_pin_direction_set; IRQ_port_direction_get; IRQ_pin_direction_get; IRQ_pin_set; IRQ_pin_get; IRQ_isr_enable; IRQ_isr_set.

☑ *RTL555* ☐ *ISR555* ☐ TBD

7. IRQ_pin_set

Description: Sets a given output pin either high or low.

Syntax: ReturnType IRQ_pin_set(*UInt8 PIOpin , boolean pin_data*);

Prototype in: ..gcc\lib\intec-lib\irq.h

Parameters:

PIOpin Desired pin to set
Range = 0 – 7
Pin 0 = 0, Pin 1 = 1, Pin 2 = 2, …, Pin 6 = 6, Pin 7 = 7

pin_data Level to set the pin.
Range = LOW or HIGH
0=GND=low 1=5V0=high

Returns: Returns a success or failure code. See ReturnTypes,(page 2) for more information.

Example:

```
#include "m_common.h"          // typedefs and some #define's for the 555
#include "irq.h"               // IRQ_init(), IRQ_port_direction_set()..
                :
IRQ_init( GPIO, GPIO );        // Set all IRQ pins for GPIO operation
                :
IRQ_pin_set( 0 , HIGH );       // Sets IRQ0 pin HIGH
IRQ_pin_set( 5 , LOW );        // Sets IRQ5 pin LOW
                :
```

Comments: GPIO pins are not continuous. Be careful that you do not mistake *IRQ6 for GPIO6 and *IRQ7 for GPIO7:

FRZ	39	40	*IRQOUT	GPIO6	39	40	GPIO7
*IRQ0	41	42	*IRQ0	GPIO0	41	42	GPIO1
*IRQ2	43	44	*IRQ3	GPIO2	43	44	GPIO3
*IRQ4	45	46	*IRQ5	GPIO4	45	46	GPIO5
*IRQ6	47	48	*IRQ7	*IRQ6	47	48	*IRQ7

Location of Pins: Schematics Sheet 7/8 CN5 Pins 39-50

Demo Programs: IRQ_gpio.ipj; IRQ_isr.ipj; IRQ_isr_all.ipj; blinkall.ipj

See Also: IRQ_init;IRQ_port_direction_set; IRQ_port_direction_get; IRQ_port_set; IRQ_port_get; IRQ_pin_direction_set; IRQ_port_direction_get; IRQ_pin_direction_get; IRQ_pin_get; IRQ_isr_enable; IRQ_isr_set.

☑ *RTL555* ☐ *ISR555* ☐ TBD

8. IRQ_port_get

Description: Gets the data from the entire 8-bit port in one function call.

Syntax: UInt8 IRQ_port_get();

Prototype in: ..gcc\lib\intec-lib\irq.h

Parameters: none

Returns: Returns an integer corresponding to the hex value of the IRQ port.
Range = 0x00– 0xFF
0=GND=low 1=5V0=high

Example:

```
#include "m_common.h"              // typedefs and some #define's for the 555
#include "irq.h"                   // IRQ_init(), IRQ_port_direction_set()..
                    :
int x;
IRQ_init( GPIO, GPIO );            // Set all IRQ pins for GPIO operation
set_IRQ_port_direction( 0x00);     // Sets all the IRQ pins for input
                    :
x=IRQ_port_get( );                 // Get the value of the IRQ port
                    :
```

Comments: GPIO pins are not continuous. Be careful that you do not mistake *IRQ6 for GPIO6 and *IRQ7 for GPIO7:

FRZ	39	40	*IRQOUT	GPIO6	39	40	GPIO7
*IRQ0	41	42	*IRQ0	GPIO0	41	42	GPIO1
*IRQ2	43	44	*IRQ3	GPIO2	43	44	GPIO3
*IRQ4	45	46	*IRQ5	GPIO4	45	46	GPIO5
*IRQ6	47	48	*IRQ7	*IRQ6	47	48	*IRQ7

The entire PIO port is read, output bits included. This means that not only can input pins be read, but the current value being driven on the output pins can also be read. Be careful to properly handle the data that is returned by this function if you are using a mix of input and output functionality on the IRQ pins.

Location of Pins: Schematics Sheet 7/8 CN5 Pins 39-50

Demo Programs: IRQ_gpio.ipj; IRQ_isr.ipj; IRQ_isr_all.ipj; blinkall.ipj

See Also: IRQ_init;IRQ_port_direction_set; IRQ_port_direction_get; IRQ_port_set; IRQ_pin_direction_set; IRQ_port_direction_get; IRQ_pin_direction_get; IRQ_pin_set; IRQ_pin_get; IRQ_isr_enable; IRQ_isr_set.

☑ *RTL555* ☐ *ISR555* ☐ TBD

9. IRQ_pin_get

Description: Gets the data from an IRQ pin.

Syntax: boolean IRQ_pin_get (UInt8 PIOpin);

Prototype in: ..gcc\lib\intec-lib\irq.h

Parameters: **IRQpin** Desired pin to get
Range = 0 - 7
Pin 0 = 0, Pin 1 = 1, Pin 2 = 2, …, Pin 6 = 6, Pin 7 = 7

Returns: **Returns a success or failure code. See ReturnTypes,(page 2) for more information.**

Example:

```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "irq.h"            // IRQ_init(), IRQ_port_direction_set()..
                    :
boolean x;
IRQ_init( GPIO, GPIO );   // Set all IRQ pins for GPIO operation
                    :
IRQ_pin_direction_set( 4 , INPUT );          // Sets the IRQ4 pin for Input
x = IRQ_pin_get( 4 );                        // Get the value of PIO4
                    :
```

Comments: GPIO pins are not continuous. Be careful that you do not mistake *IRQ6 for GPIO6 and *IRQ7 for GPIO7:

FRZ	39	40	*IRQOUT	GPIO6	39	40	GPIO7
*IRQ0	41	42	*IRQ0	GPIO0	41	42	GPIO1
*IRQ2	43	44	*IRQ3	GPIO2	43	44	GPIO3
*IRQ4	45	46	*IRQ5	GPIO4	45	46	GPIO5
*IRQ6	47	48	*IRQ7	*IRQ6	47	48	*IRQ7

The current value being driven on an output pin can also be read by this command.

Location of Pins: Schematics Sheet 7/8 CN5 Pins 39-50

Demo Programs: IRQ_gpio.ipj; IRQ_isr.ipj; IRQ_isr_all.ipj; blinkall.ipj

See Also: IRQ_init;IRQ_port_direction_set; IRQ_port_direction_get; IRQ_port_set; IRQ_port_get; IRQ_pin_direction_set; IRQ_port_direction_get; IRQ_pin_direction_get; IRQ_pin_set; IRQ_isr_enable; IRQ_isr_set.

☑ *RTL555* ☐ *ISR555* ☐ TBD

10. IRQ_isr_enable

Description: Enable/disable a particular IRQ pin.

Syntax: ReturnType IRQ_isr_enable(*UInt8 priority ,boolean enable*);

Prototype in: ..gcc\lib\intec-lib\irq.h

Parameters:

priority IRQ pin to either enable or disable the ability to interrupt.
Range = 0:7
0 = IRQ 0 1 = IRQ1 … 6 = IRQ6 7 = IRQ7

enable Eanbles or disables the ability of a given IRQ pin to interrupt
Range = ENABLED/DISABLED
0 = DISABLED , 1 = ENABLED

Returns: **Returns a success or failure code. See ReturnTypes,(page 2) for more information.**

Example:

```
#include "m_common.h"      // typedefs and some #define's for the 555
#include "irq.h"           // IRQ_init(), IRQ_port_direction_set()..
#include "ISRhandler.h"    // ISR functions

void IRQ5 ( void){
   //ISR code goes here
}
                   :
main{
                   :
   IRQ_init( IRQ, IRQ );   // Set all IRQ pins for GPIO operation
                   :
   IRQ_isr_set( 5 , IRQ5 );// Place the IRQ5 function in the  vector table
   IRQ_isr_enable( 5 , ENABLE ); // Enable IRQ5
                   :
```

Comments: Do not try to use printfs or other such statements in the ISR. The latency is too long. Set a semaphore and service the interrupt in the mainline of the program. The ISR cannot receive parameters or return data.

Location of Pins: Schematics Sheet 7/8 CN5 Pins 39-50

Demo Programs: IRQ_gpio.ipj; IRQ_isr.ipj; IRQ_isr_all.ipj; blinkall.ipj

See Also: IRQ_init;IRQ_port_direction_set; IRQ_port_direction_get; IRQ_port_set; IRQ_port_get; IRQ_pin_direction_set; IRQ_port_direction_get; IRQ_pin_direction_get; IRQ_pin_set; IRQ_pin_get; IRQ_isr_set.

☑ *RTL555* ☐ *ISR555* ☐ TBD

11. IRQ_isr_set

Description: Places a user written function into the interrupt vector table. This function will be executed when the particular IRQ pins interrupts.

Syntax: ReturnType IRQ_isr_set(*UInt8 priority* , *IRQFuncType function*);

Prototype in: ..gcc\lib\intec-lib\irq.h

Parameters:

priority IRQ pin to load with the interrupt subroutine.
Range = 0:7
0 = IRQ 0 1 = IRQ1 … 6 = IRQ6 7 = IRQ7

function Function that will be executed when the priority interrupts

Returns: **Returns a success or failure code. See ReturnTypes,(page 2) for more information.**

Example:

```
#include "m_common.h"      // typedefs and some #define's for the 555
#include "irq.h"           // IRQ_init(), IRQ_port_direction_set()..
#include "ISRhandler.h"    // ISR functions

void IRQ5 ( void){
   //ISR code goes here
}
                  :
main{
                  :
   IRQ_init( IRQ, IRQ );   // Set all IRQ pins for GPIO operation
                  :
   IRQ_isr_set( 5 , IRQ5 );          // Place IRQ5 function in the vector table
   IRQ_isr_enable( 5 , ENABLE ); // Enable IRQ5
                  :
```

COMMENTS: DO NOT TRY TO USE PRINTFS OR OTHER SUCH STATEMENTS IN THE ISR. THE LATENCY IS TOO LONG. SET A SEMAPHORE AND SERVICE THE INTERRUPT IN THE MAINLINE OF THE PROGRAM.

Location of Pins: Schematics Sheet 7/8 CN5 Pins 39-50

Demo Programs: IRQ_gpio.ipj; IRQ_isr.ipj; IRQ_isr_all.ipj; blinkall.ipj

See Also: IRQ_init;IRQ_port_direction_set; IRQ_port_direction_get; IRQ_port_set; IRQ_port_get; IRQ_pin_direction_set; IRQ_port_direction_get; IRQ_pin_direction_get; IRQ_pin_set; IRQ_pin_get; IRQ_isr_enable.

☑ *RTL555* ☐ *ISR555* ☐ TBD

11. MPIO

MIOS Parallel Input/Output

General Description

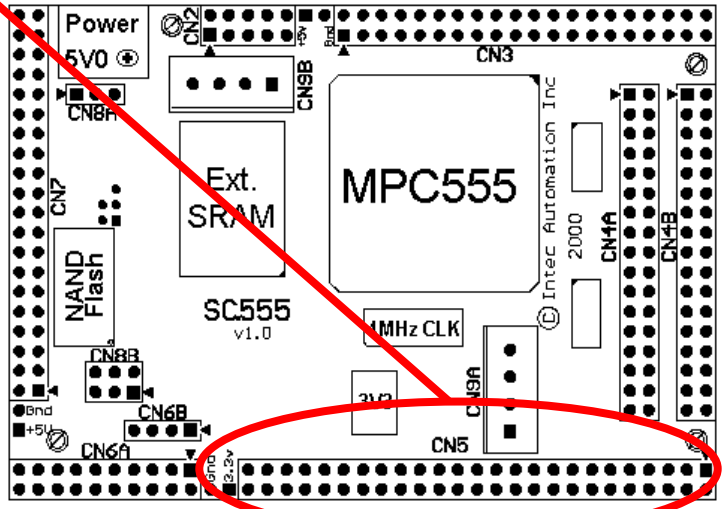
16 pin - 5 Volt General Purpose I/O port. Allows up to 16 pins to be read or written simultaneously. For more information see section 15.13 - MIOS 16-bit Parallel Port I/O Submodule (MPIOSM) in the MPC555 user's manual.

Pin Location

CN5

	1	2	
	3	4	
	5	6	
	7	8	
	9	10	
	11	12	
	13	14	
	15	16	
	17	18	
	19	20	
5V0	21	22	GND
MPIO0	23	24	MPIO1
MPIO2	25	26	MPIO3
MPIO4	27	28	MPIO5
MPIO6	29	30	MPIO7
MPIO8	31	32	MPIO9
MPIO10	33	34	MPIO11
MPIO12	35	36	MPIO13
MPIO14	37	38	MPIO15
	39	40	
	41	42	
	43	44	
	45	46	
	47	48	
	49	50	

2 x 25



Functions

Port	Pin
Initialization of the MPIO	
MPIO_port_direction_set(direction)	MPIO_pin_direction_set(MPIO pin , direction)
MPIO_port_direction_get(void)	MPIO_pin_direction_get(MPIO pin)
MPIO as an Output from the MPC555	
MPIO_port_set(data)	MPIO_pin_set (MPIO pin , pin_data)
MPIO as an Input to the MPC555	
MPIO_port_get(void)	MPIO_pin_get(MPIO pin)

1. MPIO_port_direction_set

Description: Sets the entire 16-bit port for I/O operation in one function call.

Syntax: ReturnType MPIO_port_direction_set (*UInt16 direction*);

Prototype in: ..gcc\lib\intec-lib\ss_mios.h

Parameters: **direction** 16-bit number corresponding to the I/O operation for the port.
Range = 0x0000 – 0xFFFF
0 = Input 1 = Output

Returns: Returns a success or failure code. See ReturnTypes,(page 2) for more information.

Example:

```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "ss_mios.h"        // IRQ_init(), IRQ_port_direction_set()..
                :
MPIO_port_direction_set( 0xFFFF );  // Set port direction as all outputs
                :
MPIO_port_direction_set( 0x0000 );  // Set port direction as all inputs
                :
MPIO_port_direction_set( 0x5555 );  // Set port direction as alternating I/O
                :
```

Comments: None.

Location of Pins: Schematics Sheet 7/8 CN5 Pins 23-38

Demo Programs: mpiodemo.ipj;mpoport.ipj;blinkall.ipj

See Also: MPIO Pin Direction Set; MPIO Port Direction Get; MPIO Pin Direction Get; MPIO Port Set; MPIO Pin Set; MPIO Port Get; MPIO Pin Get.

☑ *RTL555* ☐ *ISR555* ☐ TBD

---

2. MPIO_pin_direction_set

Description: Sets an individual pin to either input or output operation.

Syntax: ReturnType MPIO_pin_direction_set(UInt8 *PIOpin , boolean direction*);

Prototype in: ..gcc\lib\intec-lib\ss_mios.h

Parameters:

MPIOpin Desired pin to set the direction.
Range = 0 – 15
Pin 0 = 0, Pin 1 = 1, Pin 2 = 2, …, Pin 14 = 14, Pin 15 = 15

direction Direction to set the given pin.
Range = INPUT or OUTPUT
0 = Input 1 = Output

Returns: Returns a success or failure code. See ReturnTypes,(page 2) for more information.

Example:

```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "ss_mios.h"        // IRQ_init(), IRQ_port_direction_set()..
                    :
MPIO_pin_direction_ set( 0 , INPUT );       // Set MPIO pin 0 for input
MPIO_pin_direction_set( 1 , OUTPUT );      // Set MPIO pin 1 for output
                    :
```

Comments: None.

Location of Pins: Schematics Sheet 7/8 CN5 Pins 23-38

Demo Programs: mpiodemo.ipj;mpoport.ipj;blinkall.ipj

See Also: MPIO Port Direction Set; MPIO Port Direction Get; MPIO Pin Direction Get; MPIO Port Set; MPIO Pin Set; MPIO Port Get; MPIO Pin Get**.**

☑ *RTL555* ☐ *ISR555* ☐ TBD

3. MPIO_port_direction_get

Description: Gets the current setup of the entire 16-bit MPIO port in one function call.

Syntax: UInt16 **MPIO_port_direction_get( );**

Prototype in: ..gcc\lib\intec-lib\ss_mios.h

Parameters: none

Returns: Hexadecimal number corresponding to the pattern of I/O operation for the port.
Range = 0x0000 – 0xFFFF
0=Input 1=Output.

Example:
```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "ss_mios.h"        // IRQ_init(), IRQ_port_direction_set()..
                    :
UInt8 direction;
MPIO_port_direction_set( 0xFFFF );          // Set port direction as all outputs
                    :
direction=MPIO_port_direction_get();     // Get the current port direction
```

Comments: None.

Location of Pins: Schematics Sheet 7/8 CN5 Pins 23-38

Demo Programs: mpiodemo.ipj;mpoport.ipj;blinkall.ipj

See Also: MPIO Port Direction Set; MPIO Pin Direction Set; MPIO Pin Direction Get; MPIO Port Set; MPIO Pin Set; MPIO Port Get; MPIO Pin Get.

☑ *RTL555* ☐ *ISR555* ☐ TBD

---

4. MPIO_pin_direction_get

Description Gets the current setup of a single MPIO pin.

Syntax: boolean MPIO_pin_direction_get(UInt8 PIOpin);

Prototype in: ..gcc\lib\intec-lib\ss_mios.h

Parameters: **MPIOpin** Desired pin to get.
Range = 0 – 15
Pin 0 = 0, Pin 1 = 1, Pin 2 = 2, …, Pin 14 = 14, Pin 15 = 15

Returns: Returns the current direction of the given pin.
Range = INPUT or OUTPUT
0 = Input 1 = Output

Example:
```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "ss_mios.h"        // IRQ_init(), IRQ_port_direction_set()..
                :
boolean direction;
MPIO_port_direction_set( 0xFFFF );          // Set port direction as all outputs
                :
direction = MPIO_pin_direction_get( 5 ); // Get the direction of pin 5
```

Comments: None.

Location of Pins: Schematics Sheet 7/8 CN5 Pins 23-38

Demo Programs: mpiodemo.ipj;mpoport.ipj;blinkall.ipj

See Also: MPIO Port Direction Set; MPIO Pin Direction Set; MPIO Port Direction Get; MPIO Port Set; MPIO Pin Set; MPIO Port Get; MPIO Pin Get.

☑ *RTL555* ☐ *ISR555* ☐ TBD

---

5. MPIO_port_set

Description: Sets the data on all MPIO output pins in one command.

Syntax: ReturnType MPIO_port_set(*UInt16 data*);

Prototype in: ..gcc\lib\intec-lib\ss_mios.h

Parameters: **data** Output data to write to the MPIO port.
range = 0x0000 to 0xFFFF
0=GND=low 1=5V0=high

Returns: Returns a success or failure code. See ReturnTypes,(page 2) for more information.

Example:

```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "ss_mios.h"        // IRQ_init(), IRQ_port_direction_set()..
                :
MPIO_port_direction_set( 0xFFFF );      // Set port direction as all outputs
MPIO_port_set( 0xFFFF );                // Sets all MPIO pins HI
                :
```

Comments: Pins that are set for input are unchanged by this command.

Location of Pins: Schematics Sheet 7/8 CN5 Pins 23-38

Demo Programs: mpiodemo.ipj;mpoport.ipj;blinkall.ipj

See Also: MPIO Port Direction Set; MPIO Pin Direction Set; MPIO Port Direction Get; MPIO Pin Direction Get; MPIO Pin Set; MPIO Port Get; MPIO Pin Get.

☑ *RTL555* ☐ *ISR555* ☐ TBD

---

6. MPIO_pin_set

Description: Sets a given output pin either high or low.

Syntax: ReturnType MPIO_pin_set(*UInt8 PIOpin , boolean pin_data*);

Description: Sets the entire 16-bit port for I/O operation in one function call.

Syntax: ReturnType MPIO_port_direction_set (*direction*);

Prototype in: ..gcc\lib\intec-lib\ss_mios.h

Parameters:

PIOpin Desired pin to set
Range = 0 – 15
Pin 0 = 0, Pin 1 = 1, Pin 2 = 2, …, Pin 14 = 14, Pin 15 = 15

pin_data **L**evel to set the pin.
Range = LOW or HIGH
0 =GND = low 1 = 5V0 = high

Returns: **Returns a success or failure code. See ReturnTypes,(page 2) for more information.**

Example:

```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "ss_mios.h"        // IRQ_init(), IRQ_port_direction_set()..
                :
MPIO_port_direction_set( 0xFFFF );          // Set port direction as all outputs
MPIO_pin_set( 0 , HIGH );       // Sets MPIO0 pin HIGH
MPIO_pin_set( 5 , LOW );        // Sets MPIO5 pin LOW
                :
```

Comments: None.

Location of Pins: Schematics Sheet 7/8 CN5 Pins 23-38

Demo Programs: mpiodemo.ipj;mpoport.ipj;blinkall.ipj

See Also: MPIO Port Direction Set; MPIO Pin Direction Set; MPIO Port Direction Get; MPIO Pin Direction Get; MPIO Port Set MPIO Port Get; MPIO Pin Get.

☑ *RTL555* ☐ *ISR555* ☐ TBD

---

7. MPIO_port_get

Description: Gets the data from the entire 16-bit port in one function call.

Syntax: UInt16 **MPIO_port_get ( )**;

Prototype in: ..gcc\lib\intec-lib\ss_mios.h

Parameters: none

Returns: Returns an integer corresponding to the hex value of the MPIO port.
Range = 0x0000 – 0xFFFF
0=GND=low 1=5V0=high

Example:
```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "ss_mios.h"        // IRQ_init(), IRQ_port_direction_set()..
                    :
UInt16 port_data;
MPIO_port_direction_set( 0x0000 );          // Set port direction as all inputs
                    :
port_data = MPIO_port_get( );
                    :
```

Comments: The entire port is read, output bits included. This means that not only can input pins be read, but the current value being driven on the output pins can also be read. Be careful to properly handle the data that is returned by this function if you are using a mix of input and output functionality on the MPIO pins.

Location of Pins: Schematics Sheet 7/8 CN5 Pins 23-38

Demo Programs: mpiodemo.ipj;mpoport.ipj;blinkall.ipj

See Also: MPIO Port Direction Set; MPIO Pin Direction Set; MPIO Port Direction Get; MPIO Pin Direction Get; MPIO Port Set; MPIO Pin Set; MPIO Pin Get.

☑ *RTL555* ☐ *ISR555* ☐ TBD

---

8. MPIO_pin_get

Description: Gets the data from an MPIO pin.

Syntax: boolean MPIO_pin_get (*UInt8 MPIOpin*);

Prototype in: ..gcc\lib\intec-lib\ss_mios.h

Parameters: **MPIOpin** Desired pin to get
Range = 0 - 15
Pin 0 = 0, Pin 1 = 1, Pin 2 = 2, …, Pin 14 = 14, Pin 15 = 15

Returns: Returns a success or failure code. See ReturnTypes,(page 2) for more information.

Example:

```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "ss_mios.h"        // IRQ_init(), IRQ_port_direction_set()..
                    :
boolean pin_data;
MPIO_port_direction_set( 0x0000 );          // Set port direction as all inputs
                    :
pin_data = MPIO_pin_get( 4 );               // Get the value of PIO4
                    :
pin_data = MPIO_pin_get( 15 );              // Get the value of PIO15
                    :
```

Comments: The current value being driven on output pin can also be read by this command.

Location of Pins: Schematics Sheet 7/8 CN5 Pins 23-38

Demo Programs: mpiodemo.ipj;mpoport.ipj;blinkall.ipj

See Also: MPIO Port Direction Set; MPIO Pin Direction Set; MPIO Port Direction Get; MPIO Pin Direction Get; MPIO Port Set; MPIO Pin Set; MPIO Port Get.

☑ *RTL555* ☐ *ISR555* ☐ TBD

12. PIT

Periodic Interrupt Timer

General Information

The PIT can generates a maskable interrupt after a certain number of clock tics. The PIT can be set to generate interrupts from once every microsecond to once every 4.19 seconds. The following table shows the PIT settings, period range and resolution. For more information, see section 6.9 in the MPC555 User's Manual.

Pin Location

The PIT is an internal module, having no external pins. The user must decide on a resolution and setup the combined PIT and RTC (real time clock). The user then chooses and enables a PIT period, and interrupt level.

Clock	Division Factor	Resolution	Minimum	Maximum
OSCM	4	1µs (high)	1µs	65536µs = 0.065s
OSCM	256	64µs (low)	64µs	4194304µs ≈ 4.19s
BUCK	N/A	≈7MHz	???	???

Advanced Users

This is the formula used for the conversion of period to clock tics within the PIT_set(….) function.

Examples:

$$Period \text{ of PIT} = \frac{\text{PITC}+1}{\text{OSCM}} \cdot RTDIV$$

$$Period \text{ of PIT} = \frac{0+1}{4\text{MHz}} * 256 = 64\mu s \quad Period \text{ of PIT} = \frac{0+1}{4\text{MHz}} * 4 = 1\mu s$$

Functions

PIT / RTC combined functions	
PITRTclk_init(Clock Source , Division Factor)	sets combined PIT/RTC properties
PIT only functions	
PIT_get( )	returns time until next PIT interrupt in µs
PIT_tic_time( );	Returns the number of µs per PIT tic.
PIT_speed_set(*speed*);	Sets the resolution of the PIT.
PIT_period (*tics*);	Sets the number of PIT tics before the PIT timesout.
PIT_enable (*enable)*	Enables/disables the PIT timer
PIT Interrupt Functions	
PIT_priority_set (*priority*, *function)*	Sets the interrupt level and the PIT interrupt subroutine.
PIT_isr_set(*PIT, Error*)	Sets the PIT interrupt vector table.
PIT_isr(*void*)	Execute correct PIT isr.
PIT_interrupt_enable (*enable*);	Enables/disables the PIT interrupt.

1. PIT_get

Description: Returns the current value of the PIT timer in tics+1.

Syntax: UInt32 PIT_get( );

Prototype in: ..gcc\lib\intec-lib\timers.h

Parameters: **None**

Returns: Returns a success or failure code. See ReturnTypes,(page 2) for more information.

Example:

```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "timers.h"         //  RTC functions
                                    :
Uint32 PIT_time;
external_interrupt_enable( ENABLE );
PITRTCLK_set( OSCM , 256 );       // Must be 256 for RTC operation
 period = period / PIT_tic_time();
PIT_period( period );          // Set and enable Interrupt Priority
                                    :
PIT_time = PIT_get();
                                    :
```

Comments: The value returned by this function multiplied by PIT_tic_time() can be used to give the time in microseconds of the PIT.

Location of Pins: The PIT is an internal module, having no external pins.

Demo Programs: pit_isr.ipj

See Also: PITRTCLK_set; PIT_tic_time; PIT_speed_set; PIT_period; PIT_enable; PIT_priority_set; PIT_isr_set; PIT_isr; PIT_interrupt_enable

---

☑ *RTL555* ☐ *ISR555* ☐ TBD

2. PIT_tic_time

Description: Returns the number of microseconds per tic of the PIT timer.

Syntax: Uint16 PIT_tic_time( );

Prototype in: ..gcc\lib\intec-lib\timers.h

Parameters: **None**

Returns: The number of microseconds per tic of the PIT

Example:

```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "timers.h"         //  RTC functions
                    :
                                :
external_interrupt_enable( ENABLE );
PITRTCLK_set( OSCM , 256 );       // Must be 256 for RTC operation
 period = period / PIT_tic_time();
PIT_period( period );          // Set and enable Interrupt Priority
PIT_priority_set( 4 , PITisr ); // Set Interrupt priority
PIT_interrupt_enable( ENABLE ); // Enable the PIT interrupts
PIT_enable( ENABLE );          // Start the PIT
                    :
```

Comments:

Location of Pins: The PIT is an internal module, having no external pins.

Demo Programs: pit_isr.ipj

See Also: PITRTCLK_set; PIT_get; PIT_speed_set; PIT_period; PIT_enable; PIT_priority_set; PIT_isr_set; PIT_isr; PIT_interrupt_enable

---

☑ *RTL555* ☐ *ISR555* ☐ TBD

3. PIT_speed_set

Description: Sets the resolution of the PIT as either fast (1us) or slow(64us)

Syntax: ReturnType PIT_speed_set(*boolean speed*);

Prototype in: ..gcc\lib\intec-lib\timers.h

Parameters: **speed** Sets the resolution of the PIT
Range FAST/SLOW
FAST=1 SLOW=0

Returns: Returns a success or failure code. See ReturnTypes,(page 2) for more information.

Example:
```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "timers.h"         //  RTC functions
                                :
external_interrupt_enable( ENABLE );
PIT_speed_set( SLOW );      // Must be SLOW for RTC operation
 period = period / PIT_tic_time();
PIT_period( period );       // Set and enable Interrupt Priority
PIT_priority_set( 4 , PITisr ); // Set Interrupt priority
PIT_interrupt_enable( ENABLE ); // Enable the PIT interrupts
PIT_enable( ENABLE );       // Start the PIT
                    :
```

Comments: Setting the Pit to fast mode increases the resolution but also turns off the RTC. The speed must be set for SLOW in order to use the real time clock.

Location of Pins: The PIT is an internal module, having no external pins.

Demo Programs: pit_isr.ipj

See Also: PITRTCLK_set; PIT_get; PIT_tic_time; PIT_period; PIT_enable; PIT_priority_set; PIT_isr_set; PIT_isr; PIT_interrupt_enable

---

☑ *RTL555* ☐ *ISR555* ☐ TBD

4. PIT_period

Description: Sets the number of PIT tics before the PIT timesout.

Syntax: ReturnType PIT_period (*Uint32 tics*);

Prototype in: ..gcc\lib\intec-lib\timers.h

Parameters: **tics** Number of PIT tics before a PIT timeout
Range 0x0000 0000 to 0xFFFF FFFF

Returns: Returns a success or failure code. See ReturnTypes,(page 2) for more information.

Example:

```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "timers.h"         //  RTC functions
                                :
external_interrupt_enable( ENABLE );
PITRTCLK_set( OSCM , 256 );       // Must be 256 for RTC operation
 period = period / PIT_tic_time();
PIT_period( period );          // Set and enable Interrupt Priority
PIT_priority_set( 4 , PITisr ); // Set Interrupt priority
PIT_interrupt_enable( ENABLE ); // Enable the PIT interrupts
PIT_enable( ENABLE );          // Start the PIT
                        :
```

Comments:

Location of Pins: The PIT is an internal module, having no external pins.

Demo Programs: pit_isr.ipj

See Also: PITRTCLK_set; PIT_get; PIT_tic_time; PIT_speed_set; PIT_enable; PIT_priority_set; PIT_isr_set; PIT_isr; PIT_interrupt_enable

---

☑ *RTL555* ☐ *ISR555* ☐ TBD

5. PIT_enable

Description: Enables/disables the PIT timer.

Syntax: ReturnType PIT_enable (*boolean enable)*

Prototype in: ..gcc\lib\intec-lib\timers.h

Parameters: **enable** Enable/Disable the PIT timer.

Returns: Returns a success or failure code. See ReturnTypes,(page 2) for more information.

Example:

```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "timers.h"         //  RTC functions
                    :
external_interrupt_enable( ENABLE );
PITRTCLK_set( OSCM , 256 );       // Must be 256 for RTC operation
 period = period / PIT_tic_time();
PIT_period( period );          // Set and enable Interrupt Priority
PIT_priority_set( 4 , PITisr ); // Set Interrupt priority
PIT_interrupt_enable( ENABLE ); // Enable the PIT interrupts
PIT_enable( ENABLE );          // Start the PIT
                    :
```

Comments:

Location of Pins: The PIT is an internal module, having no external pins.

Demo Programs: pit_isr.ipj

See Also: PITRTCLK_set; PIT_get; PIT_tic_time; PIT_speed_set; PIT_period; PIT_priority_set; PIT_isr_set; PIT_isr; PIT_interrupt_enable

---

☑ *RTL555* ☐ *ISR555* ☐ TBD

6. PIT_priority_set

Description: Sets the interrupt level and the PIT interrupt subroutine.

Syntax: ReturnType PIT_priority_set (*Uint8 priority, IRQFuncType function)*

Prototype in: ..gcc\lib\intec-lib\ISRHandler.h

Parameters:

priority	Level of . Range = 0:7 0 = Highest priority 7 = lowest priority
function	Function that will be executed when the priority interrupts

Returns: Returns a success or failure code. See ReturnTypes,(page 2) for more information.

Example:

```
#include "m_common.h"   // typedefs and some #define's for the 555
#include "timers.h"          //  RTC functions
#include “ISRHandler”
                    :
external_interrupt_enable( ENABLE );
PITRTCLK_set( OSCM , 256 );       // Must be 256 for RTC operation
 period = period / PIT_tic_time();
PIT_period( period );          // Set and enable Interrupt Priority
PIT_priority_set( 4 , PIT_isr ); // Set Interrupt priority
PIT_interrupt_enable( ENABLE ); // Enable the PIT interrupts
PIT_enable( ENABLE );          // Start the PIT
                    :
}
```

Comments:

Location of Pins: The PIT is an internal module, having no external pins.

Demo Programs: pit_isr.ipj

See Also: PITRTCLK_set; PIT_get; PIT_tic_time; PIT_speed_set; PIT_period; PIT_enable; PIT_isr_set; PIT_isr; PIT_interrupt_enable

---

☐ *RTL555* ☑ *ISR555* ☐ TBD

7. PIT_isr_set – PIT Interrupt Vector Table Set

Description: Sets the vector table with the functions to run when the PIT interrupts.

Syntax: ReturnType PIT_isr_set (*IRQFuncType PIT , IRQFuncType Error*);

Prototype in: ..gcc\lib\intec-lib\ISRHandler.h

Parameters:

PIT	Function to run when a PIT interrupt occurs.
Error	Function to run when a spurious interrupt occurs.

Returns: Returns a success or failure code. See ReturnTypes (page 6) for more information.

Example:

```
#include "m_common.h"      // typedefs and some #define's for the 555
#include "timers.h"        //
#include “ISRHandler.h”
void PIT( void ){
                   :
}

main{
                   :
external_interrupt_init( defaultISR_1 );      // Setup interrupt vector table
PIT_isr_set( PIT ; defaultISR_1 );            // Setup PIT vector table
PIT_priority_set( 5 , PIT_isr );              // PIT Interrupts on LVL5
                   :
                   :
```

Comments: None.

Location of Pins: The PIT is an internal module, having no external pins.

Demo Programs: PIT_isr.ipj

See Also: PITRTCLK_set; PIT_get; PIT_tic_time; PIT_speed_set; PIT_period; PIT_enable; PIT_priority_set,; PIT_isr; PIT_interrupt_enable

---

☐ *RTL555* ☑ *ISR555* ☐ TBD

8. PIT_isr – PIT Interrupt Subroutine

Description: Determines the source of the PIT interrupt.

Syntax: void PIT_isr (*void*);

Prototype in: ..gcc\lib\intec-lib\ISRHandler.h

Parameters: None.

Returns: None.

Example:

```
#include "m_common.h"              // typedefs and some #define's for the 555
#include "timers.h"                //
void PIT( void ){
                    :
}

main{
                    :
external_interrupt_init( defaultISR_1 );     // Setup interrupt vector table
PIT_isr_set( PIT; defaultISR_1 );            // Setup PIT vector table
PIT_priority_set( 5 , PIT_isr );             // PIT Interrupts on LVL5
                    :
```

Comments: None.

Location of Pins: The PIT is an internal module, having no external pins.

Demo Programs: PIT_isr.ipj

See Also: PITRTCLK_set; PIT_get; PIT_tic_time; PIT_speed_set; PIT_period; PIT_enable; PIT_priority_set; PIT_isr_set; PIT_interrupt_enable

---

☐ *RTL555* ☑ *ISR555* ☐ TBD

9. PIT_interrupt_enable

Description: Enables/disables the PIT's ability to interrupt.

Syntax: ReturnType PIT_interrupt_enable (*boolean enable*);

Prototype in: ..gcc\lib\intec-lib\ISRHandler.h

Parameters: **tics** Number of PIT tics before a PIT timeout
Range 0x0000 0000 to 0xFFFF FFFF

Returns: Returns a success or failure code. See ReturnTypes,(page 2) for more information.

Example:

```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "timers.h"         //  RTC functions
                    :
external_interrupt_enable( ENABLE );
PITRTCLK_set( OSCM , 256 );       // Must be 256 for RTC operation
 period = period / PIT_tic_time();
PIT_period( period );         // Set and enable Interrupt Priority
PIT_priority_set( 4 , PITisr ); // Set Interrupt priority
PIT_interrupt_enable( ENABLE ); // Enable the PIT interrupts
PIT_enable( ENABLE );         // Start the PIT
                    :
```

Comments:

Location of Pins: The PIT is an internal module, having no external pins.

Demo Programs: pit_isr.ipj

See Also: PITRTCLK_set; PIT_get; PIT_tic_time; PIT_speed_set; PIT_period; PIT_enable; PIT_priority_set; PIT_isr_set; PIT_isr;

☐ *RTL555* ☑ *ISR555* ☐ TBD

---

13. PWM

Pulse Width Modulation

General Description

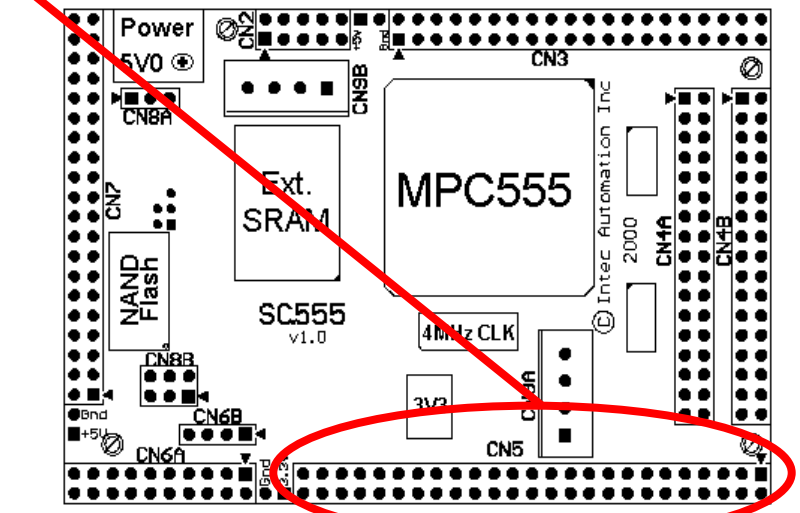
8 pin – 5 Volt Pulse Width Modulators. Can be set in either PWM mode or General Purpose Input/ Output mode. GPIO writes are not simultaneous. For interrupt information please see in this manual .See section 15.12 MIOS Pulse Width Modulation Submodule (MPWMSM) in the MPC555 user's manual.

Pin Location

CN5

	1	2	
	3	4	
	5	6	
	7	8	
	9	10	
5V0	11	12	GND
PWM0	13	14	PWM1
PWM2	15	16	PWM3
PWM4	17	18	PWM5
PWM6	19	20	PWM7
5V0	21	22	GND
	23	24	
	25	26	
	27	28	
	29	30	
	31	32	
	33	34	
	35	36	
	37	38	
	39	40	
	41	42	
	43	44	
	45	46	
	47	48	
	49	50	

2 x 25



Functions

Pulse Width Modulation Functionality	
MIOS_init(clock_prescaler)	ReturnType PWM_init(UInt8 PWMpin ,UInt8 function)
PWM_period(PWMpin, clock_prescaler , period)	ReturnType PWM_pulse(UInt8 PWMpin , UInt16 pulse)
PWM Interrupt functionality	
See _____ in this manual.	
General Purpose Input / Output Functionality	
Port	**Pin**
Initialization of the PWM for GPIO	
PWM_port_direction_set(direction)	ReturnType PWM_pin_direction_set(UInt8 PWM pin , boolean direction)
PWM_port_direction_get(void)	boolean PWM_pin_direction_get(UInt8 PWM pin)
PWM as a GPIO Output from the MPC555	
PWM_port_set(data)	ReturnType PWM_pin_set (UInt8 PWM pin , boolean pin_data)
PWM as a GPIO Input to the MPC555	
PWM_port_get(void)	boolean PWM_pin_get(UInt8 PWM pin)

1. MIOS Init

Description: Initialize the MIOS clock prescaler. This prescaler divides down f_{sys} to be used by the PWM, DA and MIOS counters.

Syntax: ReturnType MIOS_init (*UInt8 clock prescaler*);

Parameters: **clock prescaler** Number to divide f_{sys} to achieve the MIOS clock.
Range = 0 , 2 - 16 (1 is not valid)
0 = No MIOS clock, 2 = ÷2 , 3 = ÷3, … 15= ÷15, 16= ÷16

Returns: Returns a success or failure code. See Section x.x. - Return Types for more information.

Example:
```
#include "m_common.h"      // typedefs and some #define's for the 555
#include "ss_mios.h"       // IRQ_init(), IRQ_port_direction_set()..
                :
MIOS_init( 8 );                    // Divide fsys by 8
                :
```

Comments: None.

Location of Pins: Schematics Sheet 7/8 CN5 Pins 11-22

Demo Programs: PWM.ipj;

See Also: PWM_init; PWM_period; PWM_pulse; PWM_port_direction_get; PWM_port_direction_set; PWM_pin_direction_get; PWM_pin_direction_set; PWM_port_set; PWM_pin_set; PWM_port_get; PWM_pin_get; MIOS_priority_set; MIOS_isr_set; MIOS0_isr; MIOS1_isr

---

☑ *RTL555* ☐ *ISR555* ☐ TBD

2. PWM Init

Description: Initialize the PWM for either input, output, or pulse width modulation.

Syntax: ReturnType PWM_init (UInt8 PWMpin , UInt8 function);

Parameters:

PWMpin Pin whose function is to be defined.
Range = 0 - 7
Pin 0 = 0, Pin 1 = 1, Pin 2 = 2, …, Pin 6 = 6, Pin 7 = 7

Function Function to define the particular pin.
Range = INPUT , OUTPUT , PWM
0 = INPUT , 1 = OUTPUT , 2 = PWM

Returns: Returns a success or failure code. See Section x.x. - Return Types for more information.

Example:

```
#include "m_common.h"      // typedefs and some #define's for the 555
#include "ss_mios.h"       // IRQ_init(), IRQ_port_direction_set()..
                :
MIOS_init( 8 );                    // Divide f_sys by 8
PWM_init( 0 , INPUT );             // Set PWM0 for Input operation
PWM_init( 1 , OUTPUT );            // Set PWM1 for Output operation
PWM_init( 2 , PWM );               // Set PWM2 for PWM operation
```

Comments: None.

Location of Pins: Schematics Sheet 7/8 CN5 Pins 11-22

Demo Programs: PWM_gpi.ipj; PWM_gpo.ipj; PWM.ipj; blinkall.ipj

See Also: MIOS_init; PWM_period; PWM_pulse; PWM_port_direction_get; PWM_port_direction_set; PWM_pin_direction_get; PWM_pin_direction_set; PWM_port_set; PWM_pin_set; PWM_port_get; PWM_pin_get; MIOS_priority_set; MIOS_isr_set; MIOS0_isr; MIOS1_isr

---

☑ *RTL555* ☐ *ISR555* ☐ TBD

3. PWM Period

Description: Sets the period, in tics, of the PWM. For information about selecting a desired period, see Section x.x - Period and Duty Cycle Calculation in this manual.

Syntax: ReturnType PWM_period(UInt8 PWMpin, UInt16 clock_prescaler, UInt16 period);

Parameters:

PWMpin Pin whose function is to be defined.
Range = 0 - 7
Pin 0 = 0, Pin 1 = 1, Pin 2 = 2, …, Pin 6 = 6, Pin 7 = 7

Clock prescaler Divides the MIOS clock produce a clock specific to this pin.
Range = 1 - 256
1 = ÷1; 2 = ÷2; 3 = ÷ 3…255 = ÷255; 256 = ÷ 256

Period Number of tics of the PWM's clock in 1 period of the PWM.
Range = 2 – 65535
0=off, 1=Always on, 2=2 tics, 3=3 tics,…,65535=65535 tics

Returns: Returns a success or failure code. See Section x.x. - Return Types for more information.

Example:

```
#include "m_common.h"      // typedefs and some #define's for the 555
#include "ss_mios.h"       // IRQ_init(), IRQ_port_direction_set()..
                  :
MIOS_init( 8 );                    // Divide f_sys by 8
PWM_init( 1 , PWM );               // Set PWM1 for PWM operation
PWM_period( 1 , 15 , 90 );         // Set PWM1 clock for f_sys / (8*15).
                  :                // 90 tics=1 period of PWM0
```

Comments: None.

Location of Pins: Schematics Sheet 7/8 CN5 Pins 11-22

Demo Programs: PWM.ipj

See Also: MIOS_init; PWM_init; PWM_pulse; PWM_port_direction_get; PWM_port_direction_set; PWM_pin_direction_get; PWM_pin_direction_set; PWM_port_set; PWM_pin_set; PWM_port_get; PWM_pin_get; MIOS_priority_set; MIOS_isr_set; MIOS0_isr; MIOS1_isr

---

☑ *RTL555* ☐ *ISR555* ☐ TBD

---

4. PWM Pulse

Description: Sets the high time, in tics, of the PWM. This value must be less than the number of tics set in the PWM period function. For information about selecting a desired period, see Section x.x - Period and Duty Cycle Calculation in this manual.

Syntax: ReturnType PWM_pulse (*UInt8 PWMpin , UInt16 pulse*);

Parameters:

PWMpin Pin whose function is to be defined.
Range = 0 - 7
Pin 0 = 0, Pin 1 = 1, Pin 2 = 2, …, Pin 6 = 6, Pin 7 = 7

Pulse Number of tics of the specific PWM clock that the PWM=1.
Range = 1 – 65535
0=off, 1=Always on, 2=2 tics, 3=3 tics,…,65535=65535 tics

Returns: Returns a success or failure code. See Section x.x. - Return Types for more information.

Example:

```
#include "m_common.h"      // typedefs and some #define's for the 555
#include "ss_mios.h"       // IRQ_init(), IRQ_port_direction_set()..
                :
MIOS_init( 8 );                    // Divide fsys by 8
PWM_init( 1 , PWM );               // Set PWM1 for PWM operation
PWM_period( 1 , 15 , 90 );         // Set PWM1 clock for fsys / (8*15).
                                   // 90 tics=1 period of PWM0
PWM_pulse( 1 , 45 );               // Set PWM1 for 1 tic high. 45 tic / 90 tics
                :                  // 50% duty cycle
PWM_pulse( 1 , 30 );               // Set PWM1 for 30 tics high. 30 tics/90tics
                :                  // 33% duty cycle
```

Comments: None.

Location of Pins: Schematics Sheet 7/8 CN5 Pins 11-22

Demo Programs: PWM.ipj

See Also: MIOS_init; PWM_init; PWM_period; PWM_port_direction_get; PWM_port_direction_set; PWM_pin_direction_get; PWM_pin_direction_set; PWM_port_set; PWM_pin_set; PWM_port_get; PWM_pin_get; MIOS_priority_set; MIOS_isr_set; MIOS0_isr; MIOS1_isr

---

☑ *RTL555* ☐ *ISR555* ☐ TBD

5. PWM Port Direction Set

Description: Sets the entire 8-bit port for I/O operation in one function call.

Syntax: ReturnType PWM_port_direction_set (*UInt8 direction*);

Parameters: **direction** 8 bit hexadecimal number corresponding to the pattern of I/O operation for the entire port.
Range = 0x00– 0xFF
0=Input 1=Output

Returns: Returns a success or failure code. See Section x.x. - Return Types for more information.

Example:
```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "ss_mios.h"        // IRQ_init(), IRQ_port_direction_set()..
                :
PWM_port_direction_set( 0xFF );          //Set port direction as all outputs
PWM_port_direction_set( 0x00); //Set port direction as all inputs
PWM_port_direction_set( 0x55 );//Set port direction as alternating IO
                :
```

Comments: None.

Location of Pins: Schematics Sheet 7/8 CN5 Pins 11-22

Demo Programs: PWM_gpi.ipj; PWM_gpo.ipj; blinkall.ipj

See Also: MIOS_init; PWM_init; PWM_period; PWM_pulse; PWM_port_direction_get; PWM_pin_direction_get; PWM_pin_direction_set; PWM_port_set; PWM_pin_set; PWM_port_get; PWM_pin_get; MIOS_priority_set; MIOS_isr_set; MIOS0_isr; MIOS1_isr

---

☑ *RTL555* ☐ *ISR555* ☐ TBD

6. PWM Pin Direction Set

Description: Sets an individual pin to either input or output operation.

Syntax: ReturnType PWM_pin_direction_set(UInt8 PWMpin , boolean direction);

Parameters:

PWMpin Desired pin to set the direction.
Range = 0 – 7
Pin 0 = 0, Pin 1 = 1, Pin 2 = 2, …, Pin 6 = 6, Pin 7 = 7

direction Direction to set the given pin.
Range = INPUT or OUTPUT
0 = Input 1 = Output

Returns: Returns a success or failure code. See Section x.x. - Return Types for more information.

Example: :

```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "ss_mios.h"        // IRQ_init(), IRQ_port_direction_set()..
                    :
PWM_pin_direction_ set( 0 , INPUT );        // Set PWM pin 0 for input
PWM_pin_direction_set( 1 , OUTPUT );       // Set PWM pin 1 for output
                    :
```

Comments: None.

Location of Pins: Schematics Sheet 7/8 CN5 Pins 11-22

Demo Programs: PWM_gpi.ipj; PWM_gpo.ipj

See Also: MIOS_init; PWM_init; PWM_period; PWM_pulse; PWM_port_direction_get; PWM_port_direction_set; PWM_pin_direction_get; PWM_port_set; PWM_pin_set; PWM_port_get; PWM_pin_get; MIOS_priority_set; MIOS_isr_set; MIOS0_isr; MIOS1_isr

---

☑ *RTL555* ☐ *ISR555* ☐ TBD

7. PWM Port Direction Get

Description: Gets the current setup of the entire 8-bit PWM port in one function call.

Syntax: UInt8 PWM_port_direction_get( );

Parameters: None

Returns: **PortData** Hexadecimal number of the I/O operation for the port.
Range = 0x00– 0xFF
0=Input 1=Output

Example:
```
#include "m_common.h"      // typedefs and some #define's for the 555
#include "ss_mios.h"       // IRQ_init(), IRQ_port_direction_set()..
                :
boolean direction;
PWM_port_direction_set( 0xFF );   // Set port direction as all outputs
                :
x = PWM_port_direction_get (  );          // Gets the current port direction
                :
```

Comments: None.

Location of Pins: Schematics Sheet 7/8 CN5 Pins 11-22

Demo Program: PWM_gpi.ipj; PWM_gpo.ipj

See Also: MIOS_init; PWM_init; PWM_period; PWM_pulse; PWM_port_direction_set; PWM_pin_direction_get; PWM_pin_direction_set; PWM_port_set; PWM_pin_set; PWM_port_get; PWM_pin_get; MIOS_priority_set; MIOS_isr_set; MIOS0_isr; MIOS1_isr

---

☑ *RTL555* ☐ *ISR555* ☐ TBD

---

8. PWM Pin Direction Get

Description: Gets the current setup of a single PWM pin.

Syntax: boolean PWM_pin_direction_get(*UInt8 PWMpin*);

Parameters: **PWMpin** Desired pin to get.
Range = 0 – 7
Pin 0 = 0, Pin 1 = 1, Pin 2 = 2, …, Pin 6 = 6, Pin 7 = 7

Returns: Returns the current direction of the given pin.
Range = INPUT or OUTPUT
0=Input 1=Output

Example:
```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "ss_mios.h"        // IRQ_init(), IRQ_port_direction_set()..
                :
boolean direction;
MPIO_port_direction_set( 0x00 );   // Set port direction as all inputs
                :
pin_6_data = PWM_pin_direction_get( 6 ); // Get the direction of pin 6
```

Comments: None.

Location of Pins: Schematics Sheet 7/8 CN5 Pins 11-22

Demo Program: PWM_gpi.ipj; PWM_gpo.ipj

See Also: MIOS_init; PWM_init; PWM_period; PWM_pulse; PWM_port_direction_get; PWM_port_direction_set; PWM_pin_direction_set; PWM_port_set; PWM_pin_set; PWM_port_get; PWM_pin_get; MIOS_priority_set; MIOS_isr_set; MIOS0_isr; MIOS1_isr

---

☑ *RTL555* ☐ *ISR555* ☐ TBD

9. PWM Port Set

Description: Sets the data on all PWM output pins in one command.

Syntax: ReturnType set_PWM_port(UInt8 data);

Parameters: **data** Output data to write to the PWM port.
range = 0x00 to 0xFF
0=GND=low 1=5V0=high

Returns: Returns a success or failure code. See Section x.x. - Return Types for more information.

Example:
```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "ss_mios.h"        // IRQ_init(), IRQ_port_direction_set()..
                :
PWM_port_direction_set( 0xFF );  // Set port direction as all outputs
PWM_port_set( 0xFF );            // Sets all MPIO pins HI
                :
```

Comments: Pins that are set for input are unchanged by this command.

Location of Pins: Schematics Sheet 7/8 CN5 Pins 11-22

Demo Program: PWM_gpi.ipj; PWM_gpi.ipj; blinkall.ipj

See Also: MIOS_init; PWM_init; PWM_period; PWM_pulse; PWM_port_direction_get; PWM_port_direction_set; PWM_pin_direction_get; PWM_pin_direction_set; PWM_pin_set; PWM_port_get; PWM_pin_get; MIOS_priority_set; MIOS_isr_set; MIOS0_isr; MIOS1_isr

---

☑ *RTL555* ☐ *ISR555* ☐ TBD

10. PWM Pin Set

Description: Sets a given output pin either high or low.

Syntax: ReturnType PWM_pin_set(UInt8 PWMpin , boolean data);

Parameters:

PWMpin Desired pin to set
Range = 0 – 7
Pin 0 = 0, Pin 1 = 1, Pin 2 = 2, …, Pin 6 = 6, Pin 7 = 7

data Level to set the pin.
Range = LOW or HIGH
0=GND=low 1=5V0=high

Returns: Returns a success or failure code. See Section x.x. - Return Types for more information.

Example:

```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "ss_mios.h"        // IRQ_init(), IRQ_port_direction_set()..
                :
PWM_port_direction_set( 0xFF);   // Set port direction as all outputs
PWM_pin_set( 0 , HIGH );         // Sets PWM0 pin HIGH
PWM_pin_set( 5 , LOW );          // Sets PWM5 pin LOW
                :
```

Comments: None.

Location of Pins: Schematics Sheet 7/8 CN5 Pins 11-22

Demo Program: PWM_gpi.ipj; PWM_gpo.ipj

See Also: MIOS_init; PWM_init; PWM_period; PWM_pulse; PWM_port_direction_get; PWM_port_direction_set; PWM_pin_direction_get; PWM_pin_direction_set; PWM_port_set; PWM_port_get; PWM_pin_get; MIOS_priority_set; MIOS_isr_set; MIOS0_isr; MIOS1_isr

---

☑ *RTL555* ☐ *ISR555* ☐ TBD

11. PWM Port Get

Description: Gets the data from the entire 8-bit port in one function call.

Syntax: UInt8 PWM_port_get ( );

Parameters: None.

Returns: Returns an integer corresponding to the hex value of the PWM port.
Range = 0x00– 0xFF
0=GND=low 1=5V0=high

Example:
```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "ss_mios.h"          // IRQ_init(), IRQ_port_direction_set()..
                    :
UInt16 port_data;
PWM_port_direction_set( 0x00 );   // Set port direction as all inputs
                    :
port_data = PWM_port_get( );
                    :
```

Comments: The entire port is read, output bits included. This means that not only can input pins be read, but the current value being driven on the output pins can also be read. Be careful to properly handle the data that is returned by this function if you are using a mix of input and output functionality on the PWM pins.

Location of Pins: Schematics Sheet 7/8 CN5 Pins 11-22

Demo Program: PWM_gpi.ipj; PWM_gpo.ipj

See Also: MIOS_init; PWM_init; PWM_period; PWM_pulse; PWM_port_direction_get; PWM_port_direction_set; PWM_pin_direction_get; PWM_pin_direction_set; PWM_port_set; PWM_pin_set; PWM_pin_get; MIOS_priority_set; MIOS_isr_set; MIOS0_isr; MIOS1_isr

---

☑ *RTL555* ☐ *ISR555* ☐ TBD

12. PWM Pin Get

Description: Gets the data from an PWM pin.

Syntax: boolean PWM_pin_get (UInt8 PWMpin);

Inputs : **PWMpin** Desired pin to get
Range = 0 - 7
Pin 0 = 0, Pin 1 = 1, Pin 2 = 2, …, Pin 14 = 14, Pin 15 = 15

Returns: Returns data present on a specific PWM pin
Range = LOW or HIGH
0=GND=low 1=5V0=high

Example:

```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "ss_mios.h"        // IRQ_init(), IRQ_port_direction_set()..
                    :
boolean pin_data;
MPIO_port_direction_set( 0x0000 );          // Set port direction as all inputs
                    :
pin_data = PWM_pin_get( 4 );                // Get the value of PWMIO4
                    :
pin_data = PWM_pin_get( 7 );                // Get the value of PWMIO7
                    :
```

Comments: The current value being driven on output pin can also be read by this command.

Location of Pins: Schematics Sheet 7/8 CN5 Pins 11-22

Demo Program: PWM_gpi.ipj; PWM_gpo.ipj

See Also: MIOS_init; PWM_init; PWM_period; PWM_pulse; PWM_port_direction_get; PWM_port_direction_set; PWM_pin_direction_get; PWM_pin_direction_set; PWM_port_set; PWM_pin_set; PWM_port_get; MIOS_priority_set; MIOS_isr_set; MIOS0_isr; MIOS1_isr

---

☑ *RTL555* ☐ *ISR555* ☐ TBD

---

14. QADC

Queued Analog to Digital Controller

General Description

The QADC samples once every 7µs (QCLK= 2MHz) and has a 10 bit resolution. The QADC can either be setup to sample each of the 16 pins in a port once or can sample one pin in a port 64 times using continuous scan mode. The external trigger mode, and GPIO are not currently supported by the runtime library. The multiplexed analog to digital pins are not directly supported by either the runtime library or the SS555 board.

Pin Location

	CN4A	
AN0_A	1	2
AN1_A	3	4
AN2_A	5	6
AN3_A	7	8
AN4_A	9	10
AN5_A	11	12
AN6_A	13	14
AN7_A	15	16
AN8_A	17	18
AN9_A	19	20
AN10_A	21	22
AN11_A	23	24
AN12_A	25	26
AN13_A	27	28
AN14_A	29	30
AN15_A	31	32
	33	34

2x17

	CN4B	
AN0_B	1	2
AN1_B	3	4
AN2_B	5	6
AN3_B	7	8
AN4_B	9	10
AN5_B	11	12
AN6_B	13	14
AN7_B	15	16
AN8_B	17	18
AN9_B	19	20
AN10_B	21	22
AN11_B	23	24
AN12_B	25	26
AN13_B	27	28
AN14_B	29	30
AN15_B	31	32
	33	34

2x17

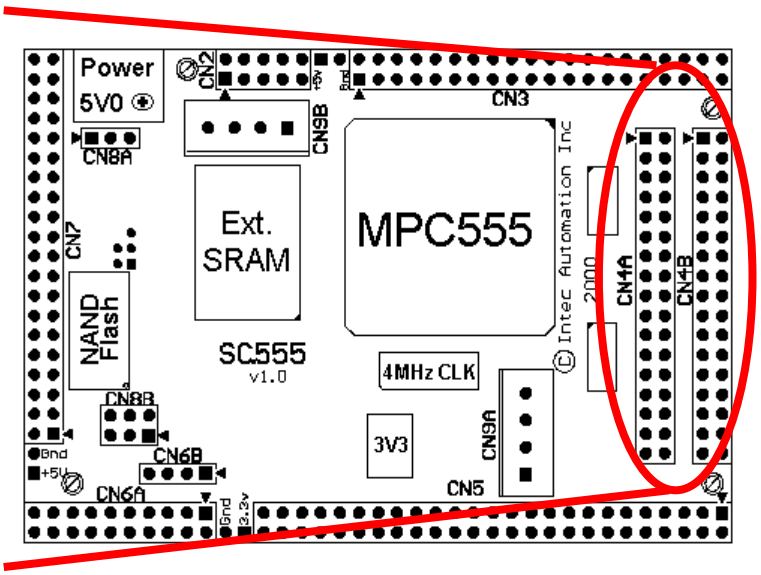


Figure 1 - Location of QADC pins

Functions

Initalization	
ADC_INIT(MODE , PORT , PIN);	
Port Sampling	**Pin Sampling**
adc_get(port, pin);	UInt16 adc_64scan_get(port);
	UInt16 adc_avg_get(port);

1. ADC_init

Description: Initializes the QADC to either scan one pin or the entire port. Both port A and port B can be set independent of each other.

Syntax: ReturnType ADC_init(UInt8 mode , boolean port , UInt16 pin);

Prototype in: ..gcc\lib\intec-lib\qadc.h

Parameters:

mode	Scan either the entire port once or a single pin 64 times. Range = SCAN_1 (scan entire port once), SCAN_64 (scan 1 pin 64 times) 0 = SCAN_1 , 1 = SCAN_64
port	Sets which of the two ports to initialize Range PORTA, PORTB 0 = PORTA 1 = PORTB
pin	Sets which pin to scan 64 times. ONLY VALID IN SCAN_64 MODE. Valid 0:15

Returns: **Returns a success or failure code. See ReturnTypes,(page 2) for more information.**

Example:

```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "qadc.h"           // IRQ_init(), IRQ_port_direction_set()..
                :
ADC_init( SCAN_1 , PORTA , 0 ); // Scan all pins of PORTA once.
                                // (0 is ignored)
ADC_init( SCAN_64 , PORTB , 4 );  // Scan pin 4 on PORTB 64 times
                :
```

Comments: The RTL555 considers the analog section in terms of pins, whereas the Motorola manual considers the analog section in terms of channels. A table is given below listing the channel number for each pin. When using the above functions the pin number should be used.

Pin #	Channel	Pin #	Channel	Pin #	Channel	Pin #	Channel
0	0	4	48	8	52	12	56
1	1	5	49	9	53	13	57
2	2	6	50	10	54	14	58
3	3	7	51	11	55	15	59

Location of Pins: Schematics Sheet 3/8 CN4a/b All Pins

Demo Programs: adc.ipj

See Also: ADC Get; ADC 64 SCAN Get; ADC avg get.

☑ *RTL555* ☐ *ISR555* ☐ TBD

---

2. ADC_get

Description: This can only be used in a port that is set for SCAN_1 mode. This returns a single reading of one of the 16 pins on a particular port.

Syntax: UInt16 adc_get(boolean port, UInt16 pin);

Prototype in: ..gcc\lib\intec-lib\qadc.h

Parameters:

port Sets which of the two ports to read
Range PORTA, PORTB
0 = PORTA 1 = PORTB

pin Sets which pin to scan 64 times.
ONLY VALID IN SCAN_64 MODE.
Valid 0:15

Returns: **Returns a success or failure code. See ReturnTypes,(page 2) for more information.**

Example:

```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "qadc.h"           // IRQ_init(), IRQ_port_direction_set()..
UInt16 adc_A15;
                :
ADC_init( SCAN_1 , PORTA , 0 ); // Scan all pins of PORTA once.
                :
adc_A15 = ADC_get( PORTA , 15 );  //Get a reading of pin 15
```

Comments: The RTL555 considers the analog section in terms of pins, whereas the Motorola manual considers the analog section in terms of channels. A table is given below listing the channel number for each pin. When using the above functions the pin number should be used.

Pin #	Channel	Pin #	Channel	Pin #	Channel	Pin #	Channel
0	0	4	48	8	52	12	56
1	1	5	49	9	53	13	57
2	2	6	50	10	54	14	58
3	3	7	51	11	55	15	59

Location of Pins: Schematics Sheet 3/8 CN4a/b All Pins

Demo Programs: adc.ipj

See Also: ADC Init; ADC 64 Scan Get; ADC avg get.

☑ *RTL555* ☐ *ISR555* ☐ TBD

---

3. ADC_64scan_get

Description: This can only be used in a port that is set for SCAN_64 mode. This returns the sum of 64 readings of a single pin.

Syntax: UInt16 adc_64scan_get(boolean port);

Prototype in: ..gcc\lib\intec-lib\qadc.h

Parameters: **port** Sets which of the two ports to read
Range PORTA, PORTB
0 = PORTA 1 = PORTB

Returns: **Returns a success or failure code. See ReturnTypes,(page 2) for more information.**

Example:
```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "qadc.h"           // IRQ_init(), IRQ_port_direction_set()..
UInt16 adc_B4;
                :
ADC_init( SCAN_64 , PORTB , 4 );  // Scan pin 4 on PORTB 64 times
                :
adc_B4=ADC_64SCAN_get(PORTB); // Get sum of 64 readings of pin 4
```

Comments: The RTL555 considers the analog section in terms of pins, whereas the Motorola manual considers the analog section in terms of channels. A table is given below listing the channel number for each pin. When using the above functions the pin number should be used.

Pin #	Channel	Pin #	Channel	Pin #	Channel	Pin #	Channel
0	0	4	48	8	52	12	56
1	1	5	49	9	53	13	57
2	2	6	50	10	54	14	58
3	3	7	51	11	55	15	59

Location of Pins: Schematics Sheet 3/8 CN4a/b All Pins

Demo Programs: adc.ipj

See Also: ADC Init; ADC Get; ADC avg get.

☑ *RTL555* ☐ *ISR555* ☐ TBD

---

4. ADC_avg_get

Description: This can only be used in a port that is set for SCAN_64 mode. This returns the sum of 64 readings of a single pin divided by 64.

Syntax: UInt16 adc_avg_get(boolean port);

Prototype in: ..gcc\lib\intec-lib\qadc.h

Parameters: **port** Sets which of the two ports to read
Range PORTA, PORTB
0 = PORTA 1 = PORTB

Returns: **Returns a success or failure code. See ReturnTypes,(page 2) for more information.**

Example:
```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "qadc.h"           // IRQ_init(), IRQ_port_direction_set()..
UInt16 adc_B4;
                :
ADC_init( SCAN_64 , PORTB , 4 );  // Scan pin 4 on PORTB 64 times
                :
adc_B4=ADC_avg_get(PORTB); // Get the avgerage of 64 readings of pin 4
                :
```

Comments: The RTL555 considers the analog section in terms of pins, whereas the Motorola manual considers the analog section in terms of channels. A table is given below listing the channel number for each pin. When using the above functions the pin number should be used.

Pin #	Channel	Pin #	Channel	Pin #	Channel	Pin #	Channel
0	0	4	48	8	52	12	56
1	1	5	49	9	53	13	57
2	2	6	50	10	54	14	58
3	3	7	51	11	55	15	59

Location of Pins: Schematics Sheet 3/8 CN4a/b All Pins

Demo Programs: adc.ipj

See Also: ADC Init; ADC Get; ADC 64 SCAN Get.

☑ *RTL555* ☐ *ISR555* ☐ TBD

15. RTC

Real Time Clock

General Information

The RTC is a counter that increments every second. The RTC can be set to interrupt every second or a number of seconds later. The RTC rolls over every 4294967296 seconds, approximately once every 136 years.

Pin Location

The RTC is an internal module, having no external pins. The user can enable or disable the RTC, Alarm interrupt, and the Once-per-Second interrupt. The user can choose to change the interrupt level and the alarm time.

Advanced Users

This is how the MPC555 calculates the one-second real time clock counter using a 4MHz crystal. See Figure 6-6 in the MPC555 User's Manual for more information.

$$1\text{second} = \frac{\text{PITRTCLK}}{15625} = \frac{\frac{OSCM}{256}}{15625} = \frac{4MHz}{256*15625} = \frac{4MHz}{4000000} = 1Hz$$

$$1\text{second} \neq \frac{\frac{OSCM}{4}}{15625} = \frac{4MHz}{4*15625} = \frac{4MHz}{62500} = 64Hz = 15.6ms$$

The division factor of the PITRTCLK must be set to 256 to properly generate a 1second RTC.

Functions

PIT / RTC combined functions	
PITRTCLK_set(Clock Source , **256**)	sets combined PIT/RTC properties Must be 256 for RTC operation
RTC only functions	
RTC_get( );	Returns current value of the PIT
RTC_write(*RTC_value*);	Sets the current value, in seconds, of the RTC.
RTC Interrupt Functions	
RTC_priority_set(*priority; function*);	Setup the interrupt for the RTC alarm.
RTC_isr_set(*Alarm, OncePerSecond , Error*)	Sets the RTC interrupt vector table.
void RTC_isr(*void*)	Execute correct RTC isr.
RTC_sec_int(*SEC_enable*);	Enables/disables the once per second interrupt.
RTC_sec_int(*ALR_enable; RTC_alarm*);	Setup the once RTC timer alarm interrupt.

1. RTC_get

Description: Returns the current value, in seconds, of the RTC timer.

Syntax: UInt32 RTC_get( );

Prototype in: ..gcc\lib\intec-lib\timers.h

Parameters: **None**

Returns: Returns a success or failure code. See ReturnTypes,(page 2) for more information.

Example:

```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "timers.h"         //  RTC functions
Uint32 RTC_time;
                    :
RTC_time = RTC_get();
RTC_time= RTC_time + 60;   //Interrupt in 1 minute
RTC_alr_int( ENABLE , RTC_time ) //Set alarm interrupt
RTC_priority_set( 5 , RTCisr );// Interrupts on LVL5
RTC_enable( ENABLE );       // Start the RTC
                    :
```

Comments: None.

Location of Pins: The RTC is an internal module, having no external pins.

Demo Programs: rtc_isr.ipj

See Also: PITRTCLK_set; RTC_write; RTC_sec_int; RTC_alr_int; RTC_priority_set;

☑ *RTL555* ☐ *ISR555* ☐ TBD

2. RTC_write

Description: Sets the current value, in seconds, of the RTC timer.

Syntax: ReturnType RTC_write(*UInt32 RTC_value*);

Prototype in: ..gcc\lib\intec-lib\timers.h

Parameters: **RTC_value** Current value to load into the RTC timer.

Returns: Returns a success or failure code. See ReturnTypes,(page 2) for more information.

Example:

```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "timers.h"         //  RTC functions
                    :
RTC_write( 0 );             // Reset the RTC
RTC_time = RTC_get();
RTC_time= RTC_time + 60;   //Interrupt in 1 minute
RTC_alr_int( ENABLE , RTC_time ) //Set alarm interrupt
RTC_priority_set( 5 , RTCisr );// Interrupts on LVL5
RTC_enable( ENABLE );       // Start the RTC
                    :
```

Comments: None.

Location of Pins: The RTC is an internal module, having no external pins.

Demo Programs: rtc_isr.ipj

See Also: PITRTCLK_set;RTC_get; RTC_sec_int; RTC_alr_int; RTC_priority_set;

☑ *RTL555* ☐ *ISR555* ☐ TBD

3. RTC_priority_set – RTC Interrupt Priority Set

Description: Setup the interrupt for the RTC alarm.

Syntax: ReturnType RTC_priority_set(*UInt8 priority; IRQFuncType function*);

Prototype in: ..gcc\lib\intec-lib\timers.h

Parameters:

priority	Priority level for the RTC alarm to interrupt on. **Range**: 0..7
function	Function to run when the RTC interrupts.

Returns: Returns a success or failure code. See ReturnTypes,(page 2) for more information.

Example:

```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "timers.h"         //  RTC functions
                    :
RTC_time = RTC_get();
RTC_time= RTC_time + 60;   //Interrupt in 1 minute
RTC_sec_int( ENABLE) //Set the once per second interrupt
RTC_priority_set( 5 , RTCisr );// Interrupts on LVL5
RTC_enable( ENABLE );       // Start the RTC
                    :
```

Comments: There is only one interrupt level specified for the RTC. The function must determine which of the two interrupt sources generated the interrupt.

Location of Pins: The RTC is an internal module, having no external pins.

Demo Programs: rtc_isr.ipj

See Also: PITRTCLK_set;RTC_get; RTC_write; RTC_sec_int; RTC_alr_int;

☐ *RTL555* ☑ *ISR555* ☐ TBD

4. RTC_isr_set – RTC Interrupt Vector Table Set

Description: Sets the vector table with the functions to run when the RTC interrupts.

Syntax: ReturnType RTC_isr_set (*IRQFuncType Alarm, IRQFuncType OncePerSecond, IRQFuncType Error*);

Prototype in: ..gcc\lib\intec-lib\ISRHandler.h

Parameters:

Alarm	Function to run when the RTC counter is equal to the alarm register.
OncePerSecond	Function to run after each RTC tic.
Error	Function to run when a spurious interrupt occurs.

Returns: Returns a success or failure code. See ReturnTypes (page 6) for more information.

Example:

```
#include "m_common.h"      // typedefs and some #define's for the 555
#include "timers.h"        //
#include “ISRHandler.h”
void RTC_alarm( void ){
                   :
}
void RTC_sec( void ){
                   :
}

main{
                   :
external_interrupt_init( defaultISR_1 );      // Setup interrupt vector table
RTC_isr_set( RTC_alarm, RTC_sec , defaultISR_1 );//Set RTC vector table
RTC_priority_set( 5 , RTC_isr );              // RTC Interrupts on LVL5
                   :
                   :
```

Comments: None.

Location of Pins: The RTC is an internal module, having no external pins.

Demo Programs: RTC_isr.ipj

See Also:

☐ *RTL555* ☑ *ISR555* ☐ TBD

5. RTC_isr – RTC Interrupt Subroutine

Description: Determines the source of the RTC interrupt.

Syntax: void RTC_isr (*void*);

Prototype in: ..gcc\lib\intec-lib\ISRHandler.h

Parameters: None.

Returns: None.

Example:

```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "timers.h"         //
#include “ISRHandler.h”
void RTC_alarm( void ){
                    :
}
void RTC_sec( void ){
                    :
}

main{
                    :
external_interrupt_init( defaultISR_1 );        // Setup interrupt vector table
RTC_isr_set( RTC_alarm, RTC_sec , defaultISR_1 );//Set RTC vector table
RTC_priority_set( 5 , RTC_isr );            // RTC Interrupts on LVL5
                    :
```

Comments: None.

Location of Pins: The RTC is an internal module, having no external pins.

Demo Programs: RTC_isr.ipj

See Also:

☐ *RTL555* ☑ *ISR555* ☐ TBD

6. RTC_sec_int – RTC Once-Per-Second Interrupt

Description: Enables or disables the once per second interrupt on the RTC timer.

Syntax: ReturnType RTC_sec_int(*boolean SEC_enable*);

Prototype in: ..gcc\lib\intec-lib\ISRHandler.h

Parameters: **SEC_enable** Enable/Disable the once-per-second interrupt.

Returns: Returns a success or failure code. See ReturnTypes,(page 2) for more information.

Example:

```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "timers.h"         //  RTC functions
                    :
RTC_time = RTC_get();
RTC_time= RTC_time + 60;   //Interrupt in 1 minute
RTC_sec_int( ENABLE) //Set the once per second interrupt
RTC_priority_set( 5 , RTCisr );// Interrupts on LVL5
RTC_enable( ENABLE );       // Start the RTC
                    :
```

Comments: None.

Location of Pins: The RTC is an internal module, having no external pins.

Demo Programs: rtc_isr.ipj

See Also: PITRTCLK_set;RTC_get; RTC_write; RTC_alr_int; RTC_priority_set;

☐ *RTL555* ☑ *ISR555* ☐ TBD

7. RTC_alr_int – RTC Alarm Interrupt

Description: Setup the once RTC timer alarm interrupt.

Syntax: ReturnType RTC_sec_int(*boolean ALR_enable; UInt64 RTC_alarm*);

Prototype in: ..gcc\lib\intec-lib\ISRHandler.h

Parameters: **ALR_enable** Enable/Disable the alarm interrupt.

RTC_alarm Time to set the RTC to interrupt.

Returns: Returns a success or failure code. See ReturnTypes,(page 2) for more information.

Example:

```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "timers.h"         //  RTC functions
                    :
RTC_time = RTC_get();
RTC_time= RTC_time + 60;   //Interrupt in 1 minute
RTC_alr_int( ENABLE , RTC_time ) //Set alarm interrupt
RTC_priority_set( 5 , RTCisr );// Interrupts on LVL5
RTC_enable( ENABLE );         // Start the RTC
                    :
```

Comments: The RTC, by default, is running. This means that the initial value of the RTC cannot be guaranteed to be zero. Either set the RTC to a known state or base the alarm interrupt on the current value of the RTC plus some offset.

Location of Pins: The RTC is an internal module, having no external pins.

Demo Programs: rtc_isr.ipj

See Also: PITRTCLK_set;RTC_get; RTC_write; RTC_sec_int; RTC_priority_set;

☐ *RTL555* ☑ *ISR555* ☐ TBD

```
ReturnType                  // See ReturnType in the SS555 Manual for codes
   RTC_enable(
      boolean enable );
```

```
ReturnType                  // See ReturnType in the SS555 Manual for codes
   PITRTCLK_set(            // Sets the PIT and RTC clock
      boolean source,       // source for the PIT and RTC
                            // OSCM or BUCK
      UInt16 RTDIV);        // PIT/RT clock divider
                            // 4   PIT = 1 us to 65536 us in 1 us increments
                            // 256 PIT = 64 us to 4.19s in 64 us increments
```

16. SCI

Seral Communications Interface

General Description

The SCI ports are typically controlled with the commands associated with the stdio.h library. These commands include printf, scanf, and many others. The SS555 also allows base functions that can be used for hardware handshaking on the SCI1 port. This section details the hardware handshaking functions. A description of the stdio.h library can be found in the gcc documentation.

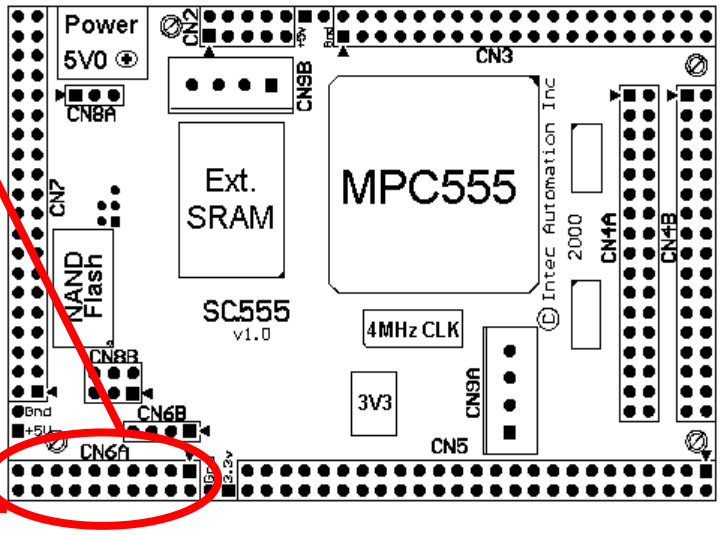
Pin Location

CN7

	20	19
	18	17
PC_CTS	16	15
PC_RTS	14	13
	12	11
	10	9
	8	7
	6	5
	4	3
	2	1

PC_DTR

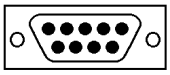
2 x 0



PC_DTR becomes *SER_RST SER_ACK becomes PC_CTS PC_RTS becomes SER_INT

Functions

From PC to SS555
ser_int_en(enable)
ser_rst_en(enable)
From SS555 to PC
ser_ack(ack)



1. ser_int_en - Serial Interrupt Enable

Description: Enabe/Disable the PC from being able to interrupt the SS555. If the PC holds PC_RTS high, IRQ0, the non-maskable interrupt, on the SS555 is asserted.

Syntax: ReturnType ser_int_en (*boolean enable*);

Prototype in: ..gcc\lib\intec-lib\cpld.h

Parameters: **enable** Enables/Disables the PC ability to interrupt the SS555.
Range = ENABLE, DISABLE
DISABLE = 0 ENABLE = 1

Outputs : Returns a success or failure code. See Section x.x. - Return Types for more information.

Example:

```
#include "m_common.h"      // typedefs and some #define's for the 555
#include "cpld.h"          // ser_int_en();ser_rst_en;ser_ack()
                :
ser_int_en( ENABLE );    // Enable COM1 to interrupt the MPC555
                :
ser_int_en( DISABLE );   // Disable COM1 to interrupt the MPC555
                :
```

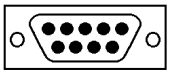
Comments: None.

Location of Pins: Schematics Sheet 5/8 CN6A Pin 14

Demo Programs: ser_int.ipj

See Also: serial reset enable; serial acknowledge.

☑ *RTL555* ☐ *ISR555* ☐ TBD



2. ser_rst_en - Serial Reset Enable

Description: Enabe/Disable the PC from being able to assert the asynchronous *HRESET on the SS555. If the PC holds PC_DTR low, *HRESET (hardware reset) on the SS555 is asserted.

Syntax: ReturnType ser_int_en (*boolean enable*);

Prototype in: ..gcc\lib\intec-lib\cpld.h

Parameters: **enable** Enables/Disables the PC ability to interrupt the SS555.
Range = ENABLE, DISABLE
DISABLE = 0 ENABLE = 1

Returns: : Returns a success or failure code. See Section x.x. - Return Types for more information.

Example:

```
#include "m_common.h"      // typedefs and some #define's for the 555
#include "cpld.h"          // ser_int_en();ser_rst_en;ser_ack()
                :
ser_rst_en( ENABLE );    // Enable COM1 to reset the MPC555
ser_rst_en( DISABLE );   // Disable COM1 to reset the MPC555
                :
```

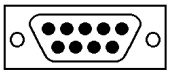
Comments: None.

Location of Pins: Schematics Sheet 5/8 CN6A Pin 17

Demo Programs: ser_rst.ipj

See Also: serial interrupt enable; serial acknowledge.

☑ *RTL555* ☐ *ISR555* ☐ TBD



3. Serial Acknowledge

Description: Sets/Clears the SER_ACK pin on the SS555. This sets/clears the PC_CTS line on the PC.

Syntax: ReturnType ser_ack(*boolean acknowledge*);

Prototype in: ..gcc\lib\intec-lib\cpld.h

Parameters: **acknowledge** clears the acknowledge signal to the PC.
Range = 0,1

Returns: Returns a success or failure code. See Section x.x. - Return Types for more information.

Example:
```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "cpld.h"           // ser_int_en();ser_rst_en;ser_ack()
                :
ser_ack( 1 );               // Set Serial Acknowledge high.
ser_ack( 0 );               // Ser Serial Acknowledge low.
                :
```

Comments: None

Location of Pins: Schematics Sheet 5/8 CN6A Pin 16

Demo Programs: ser_ack.ipj

See Also: serial interrupt enable; serial reset enable

☑ *RTL555* ☐ *ISR555* ☐ TBD

17. TB

Time Base

General Information

The TB can generate a maskable interrupt after the time base counter reaches the value programmed into one of two reference registers. The TB and DEC timers both have a common clock source. The TB can be set to generate interrupts from once every microsecond to approximately once every 13 hours. The TB is not affected by HRESET, and therefore will hold its value after a debugger initialized reset and will continue to operate in low power modes. For more information, see section 6.7 in the MPC555 User's Manual.

Pin Location

The TB is an internal module, having no external pins.

Advanced Users

See the Advanced User's section of the DEC for information on using the TMBCLK run from the system frequency.

Functions

TB / DEC combined functions	
TMBCLK_set(TBS)	Sets source of TB/DEC clock
TMBCLK_en(TMB_enable)	Enables the TMB clock
TB only functions	
TB_set(tb)	sets the current value of the TB
TB_get( )	returns the current value of the TB
TB Interrupt Functions	
TB_priority_set(priority , function);	Setup of TB interrupt
TB_isr_set(Ref0 , Ref1 , Error)	Setup of TB interrupt vector table
TB_isr(void)	Execute correct TB isr.
TB_ref_set(ref , tbref, enable)	Setup of TB reference registers to interrupt

1. TB_set – Time Base Set

Description: Set the current value of the time base register.

Syntax: ReturnType TB_set (*UInt64 tb*);

Prototype in: ..gcc\lib\intec-lib\timers.h

Parameters: **tb** New time to setup the time base.
Due to constraints on variable size, this is equal to time-1.
Range = 0 – 0xFFFF FFFF FFFF FFFF

Returns: Returns a success or failure code. See Section x.x. - Return Types for more information.

Example:
```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "timers.h"         //
                    :
TMBCLK_set( OSCM );   // Time base clock source=oscillator (4MHz)
TB_set( 0x1000 );           //  Set the time base to a value of 0x1000
                    :
```

Comments: None.

Location of Pins: The TB is an internal module, having no external pins.

Demo Programs: tb_isr.ipj

See Also: TMBCLK_set; TB_get; TB_ref_set; TB_priority_set; TB_isr_set; TB_isr; TB_ref_set

☑ *RTL555* ☐ *ISR555* ☐ TBD

2. TB_get– Time Base Get

Description: Gets the current value of the time base register.

Syntax: UInt64 TB_get ( );

Prototype in: ..gcc\lib\intec-lib\timers.h

Parameters: None.

Returns: Returns the current value of the time base register. Adding one to this value gives the correct number of time base tics that have elapsed.

Example:

```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "timers.h"         //
                    :
Uint64 TBtime;
TMBCLK_set( OSCM );    // TB clock is the crystal oscillator (4MHz)
TB_time=TB_get( );          //  Get the time base value
                    :
```

Comments: None.

Location of Pins: The TB is an internal module, having no external pins.

Demo Programs: tb_isr.ipj

See Also: TMBCLK_set; TB_set; TB_ref_set; TB_priority_set; TB_isr_set; TB_isr; TB_ref_set

---

☑ *RTL555* ☐ *ISR555* ☐ TBD

3. TB_priority_set – Time Base Interrupt Priority Setup

Description: Setup the time base interrupt level and assign a time base interrupt subroutine.

Syntax: ReturnType TB_priority_set(UInt8 priority , IRQFuncType function)

Prototype in: ..gcc\lib\intec-lib\timers.h

Parameters:

priority Interrupt priority level of the time base.
Range: 0 – 7 0 – high priority 7 – low priority

Function Interrupt subroutine function to run whenever a TB interrupt occurs.

Returns: Returns a success or failure code. See Section x.x. - Return Types for more information.

Example:

```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "timers.h"
#include “ISRHandler.h”
IRQFuncType TB_ref0( void ){
                    :
}
IRQFuncType TB_ref1( void ){
                    :
}
                    :
external_interrupt_init( defaultISR_2 );        // Fill table with given function
external_interrupt_enable( ENABLE );            // Enable the interrupt controller
TMBCLK_set( OSCM );                             // Set TB clock source
TB_ref_set( 0 , timebase0 , ENABLE );           // Setup TB reference 0
TB_ref_set( 1 , timebase1 , ENABLE );           // Setup TB reference 1
TB_isr_set( TB_ref0 , TB_ref1 , defaultISR_2 ); // Setup TB vector table
available = ISR_is_empty( 7 , defaultISR_2 ); // Check Priority 7 interrupt
if ( available ){
   TB_priority_set( 7 , TB_isr );               // TB_isr to determine TB source
}
TB_set( 1 );                                    // Restart the TB clock
 TMBCLK_en( ENABLE );                           // Start the Time Base
                    :
```

Comments: None.

Location of Pins:

Demo Programs: tb_isr.ipj

See Also: TMBCLK_set; TB_set; TB_get; TB_ref_set; TB_isr_set; TB_isr; TB_ref_set

---

☐ *RTL555* ☑ *ISR555* ☐ TBD

4. TB_isr_set – TB Interrupt Vector Table Set

Description: Sets the vector table with the functions to run when the TB interrupts.

Syntax: ReturnType TB_isr_set (*IRQFuncType Alarm, IRQFuncType OncePerSecond, IRQFuncType Error*);

Prototype in: ..gcc\lib\intec-lib\ISRHandler.h

Parameters:

Alarm	Function to run when the TB counter is equal to the alarm register.
OncePerSecond	Function to run after each TB tic.
Error	Function to run when a spurious interrupt occurs.

Returns: Returns a success or failure code. See ReturnTypes (page 6) for more information.

Example:

```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "timers.h"
#include “ISRHandler.h”
IRQFuncType TB_ref0( void ){
                    :
}
IRQFuncType TB_ref1( void ){
                    :
}
                    :
external_interrupt_init( defaultISR_2 );      // Fill table with given function
external_interrupt_enable( ENABLE );          // Enable the interrupt controller
TMBCLK_set( OSCM );                           // Set TB clock source
TB_ref_set( 0 , timebase0 , ENABLE );         // Setup TB reference 0
TB_ref_set( 1 , timebase1 , ENABLE );         // Setup TB reference 1
TB_isr_set( TB_ref0 , TB_ref1 , defaultISR_2 ); // Setup TB vector table
TB_priority_set( 7 , TB_isr );                // TB_isr to determine TB source
TB_set( 1 );                                  // Restart the TB clock
 TMBCLK_en( ENABLE );                         // Start the Time Base
                    :
```

Comments: None.

Location of Pins: The TB is an internal module, having no external pins.

Demo Programs: TB_isr.ipj

See Also: TMBCLK_set; TB_set; TB_get; TB_ref_set; TB_priority_set; TB_isr; TB_ref_set

---

☐ *RTL555* ☑ *ISR555* ☐ TBD

5. TB_isr – TB Interrupt Subroutine

Description: Determines the source of the TB interrupt.

Syntax: void TB_isr (*void*);

Prototype in: ..gcc\lib\intec-lib\ISRHandler.h

Parameters: None.

Returns: None.

Example:

```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "timers.h"
#include “ISRHandler.h”
IRQFuncType TB_ref0( void ){
                    :
}
IRQFuncType TB_ref1( void ){
                    :
}
                    :
external_interrupt_init( defaultISR_2 );        // Fill table with given function
external_interrupt_enable( ENABLE );            // Enable the interrupt controller
TMBCLK_set( OSCM );                             // Set TB clock source
TB_ref_set( 0 , timebase0 , ENABLE );           // Setup TB reference 0
TB_ref_set( 1 , timebase1 , ENABLE );           // Setup TB reference 1
TB_isr_set( TB_ref0 , TB_ref1 , defaultISR_2 ); // Setup TB vector table
available = ISR_is_empty( 7 , defaultISR_2 ); // Check Priority 7 interrupt
if ( available ){
   TB_priority_set( 7 , TB_isr );               // TB_isr to determine TB source
}
TB_set( 1 );                                    // Restart the TB clock
 TMBCLK_en( ENABLE );                           // Start the Time Base
                    :
```

Comments: None.

Location of Pins: The TB is an internal module, having no external pins.

Demo Programs: TB_isr.ipj

See Also: TMBCLK_set; TB_set; TB_get; TB_ref_set; TB_priority_set; TB_isr_set; TB_ref_set

---

☐ *RTL555* ☑ *ISR555* ☐ TBD

6. TB_ref_set – Time Base Reference Setup

Description: Setup of one of the two time base reference registers.

Syntax: ReturnType TB_ref_set(boolean ref , UInt32 tbref , boolean enable);

Prototype in: ..gcc\lib\intec-lib\timers.h

Parameters:

ref	Which of the two references to program Range: 0=REF0 1=REF1
tbref	value to program into the time base reference register Range: 0x0000 0000 – 0xFFFF FFFF
Enable	Enable or disable the current time base reference register

Returns: Returns a success or failure code. See Section x.x. - Return Types for more information.

Example:

```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "timers.h"
#include “ISRHandler.h”
IRQFuncType TB_ref0( void ){
                    :
}
IRQFuncType TB_ref1( void ){
                    :
}
                    :
external_interrupt_init( defaultISR_2 );      // Fill table with given function
external_interrupt_enable( ENABLE );          // Enable the interrupt controller
TMBCLK_set( OSCM );                           // Set TB clock source
TB_ref_set( 0 , timebase0 , ENABLE );         // Setup TB reference 0
TB_ref_set( 1 , timebase1 , ENABLE );         // Setup TB reference 1
TB_isr_set( TB_ref0 , TB_ref1 , defaultISR_2 ); // Setup TB vector table
TB_priority_set( 7 , TB_isr );                // TB_isr to determine TB source
TB_set( 1 );                                  // Restart the TB clock
TMBCLK_en( ENABLE );                          // Start the Time Base
                    :
```

Comments: None.

Location of Pins: The TB is an internal module, having no external pins.

Demo Programs: tb_isr.ipj

See Also: TMBCLK_set; TB_set; TB_get; TB_ref_set; TB_priority_set; TB_isr_set; TB_isr;

☐ *RTL555* ☑ *ISR555* ☐ TBD

18. Timing

General Description

The MPC555 can be run at many different operating frequencies. Reducing the operating frequency reduces the current consumed and the noise generated by the MPC555. Although there are many combinations of operating speeds that are possible, the MPC555 library limits the choices and reconfigures the SCI ports and the ADC to allow for seamless transition between operating speeds. It should be noted that though the SCI and ADC will adjust to continue communicating at the same speed before the change in operating frequency, the CAN, PWM and DA pins do not reconfigure themselves. The period/baud rate will change on these modules if the frequency is adjusted

Pin Location

There are no external pins for the timing section.

Functions

Timing
sysclk_set(freq_select);

1. sysclk_set

Description: Sets the system frequency and adjusts the SCI baud rate and the ADC sampling rate.

Syntax: ReturnType sysclk_set(int freq_select);

Prototype in: ..gcc\lib\intec-lib\timing.h

Parameters: **freq_select** Sets the frequency of the MPC555.
Range kHz_62_5, kHz_250, MHz_1, MHz_16, MHz_20, MHz_32, MHz_40
0=kHz_62_5, 1=kHz_250, 2=MHz_1, 3=MHz_16, 4=MHz_20, 5=MHz_32, 6=MHz_40

Returns: Returns a success or failure code. See Section x.x. - Return Types for more information.

Example:
```
#include "m_common.h"       // typedefs and some #define's for the 555
#include "timing.h"         // sysclk_set()
                    :
sysclk_set( MHz_40 );   // Run the MPC555 at 40 MHz
                    :
sysclk_set( MHz_32 );   // Run the MPC555 at 32 MHz
```

Comments:

Location of Pins: None.

Demo Programs: uart1.ipj; uart2.ipj

See Also:

☑ *RTL555* ☐ *ISR555* ☐ TBD
