# Reference Manual

**iMOTION™ motor control IC for single motor drive**

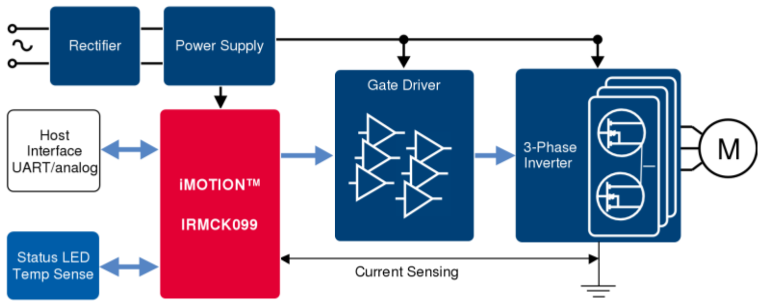
**Quality Requirement Category: Industry**

## Features

- Ready-to-use solution for high efficiency variable speed drives
- Pre-programmed motion control engine (MCE)
- Sensor less field oriented control (FOC) of permanent magnet synchronous motors (PMSM)
- Support for up to 31 sets of motor parameters
- Current measurement based on single or leg shunt
- Integrated oscillator,  A/D converter, OP amps & comparators
- Integrated protection features
- Package: 5x5mm$^2$ QFN-32

## Applications

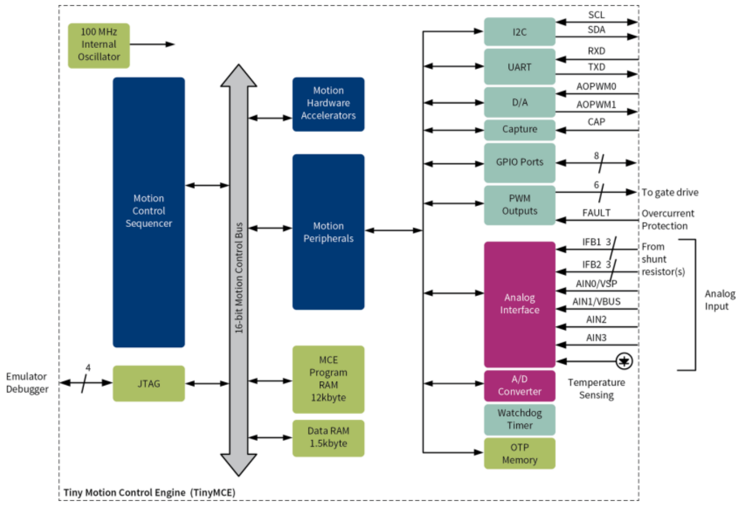
- Pumps & fans
- Drones, multicopters
- Home appliances
- Any other PMSM drive



## Description

The IRMCK099 combines the iMOTION™ motor control engine (MCE) with all peripherals required to realize a complete variable speed drive. The IRMCK099 does not require algorithm programming and can be used in combination with a µIPM™ or a discrete power stage.

The MCE implements sensor less field oriented control using single or leg shunt current feedback and uses space vector PWM with sinusoidal signals to provide highest energy efficiency.



# Table of Contents

# 1 Overview

IRMCK099 is designed to implement high performance control solutions for advanced inverterized appliance and most fan/ pump motor control. IRMCK099 contains the flexible Tiny Motion Control Engine (TinyMCE) for sensorless control of permanent magnet motors over the full speed range. The TinyMCE implements sensorless Field Oriented Control using single or leg shunt current feedback by a combination of hardware and IR-supplied firmware elements. Key components of the complex sensorless control algorithms, such as the Angle Estimator, are provided as complete pre-defined control blocks. The ASIC is designed to eliminate external components and reduce cost by including an A/D converter, analog amplifiers, an overcurrent comparator, watchdog timer and internal oscillator. Strong startup and configuration tools get the motor running quickly without any programming. A standby power mode can help to increase overall system efficiency. The motor can be either a permanent magnet motor or an induction motor depending on application needs, with a common control structure: Sensorless Field Oriented Control. **Figure 1** shows a typical application schematic of sensorless control with single shunt current sense resistor.

The IRMCK099 devices contain 16K bytes of One Time Programmable (OTP) memory and 12K bytes RAM for instruction storage and 3K bytes for data RAM.

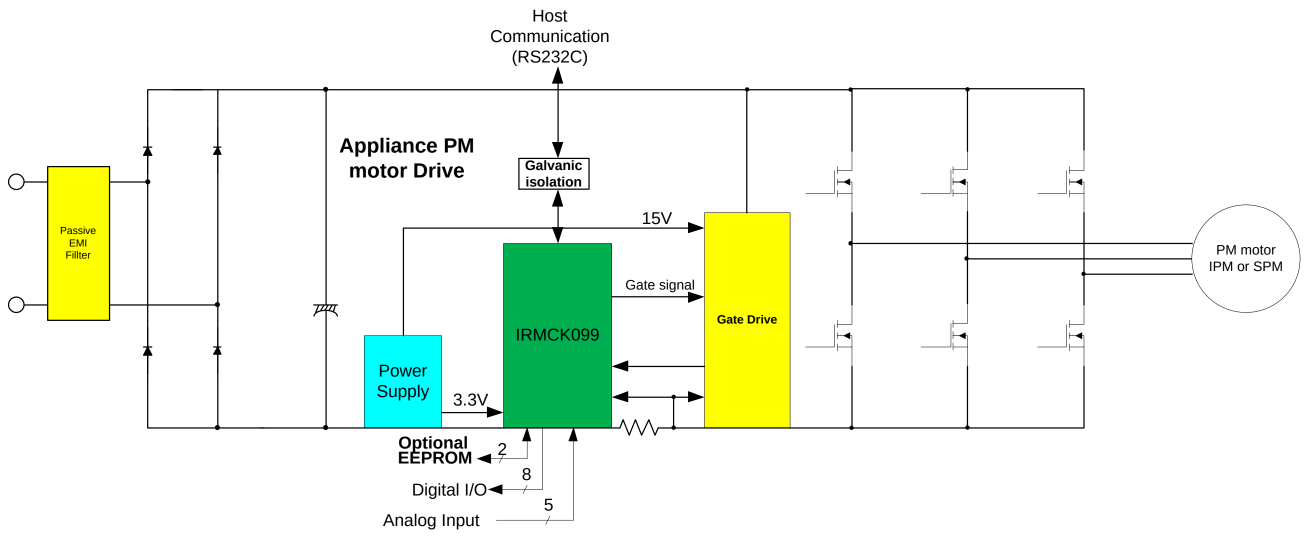


**Figure 1. Typical Application Diagram Using IRMCK099**

A block diagram of the IRMCK099 is shown in **Figure 2**. It is available in a compact 5mm x 5mm 32-pin QFN package.

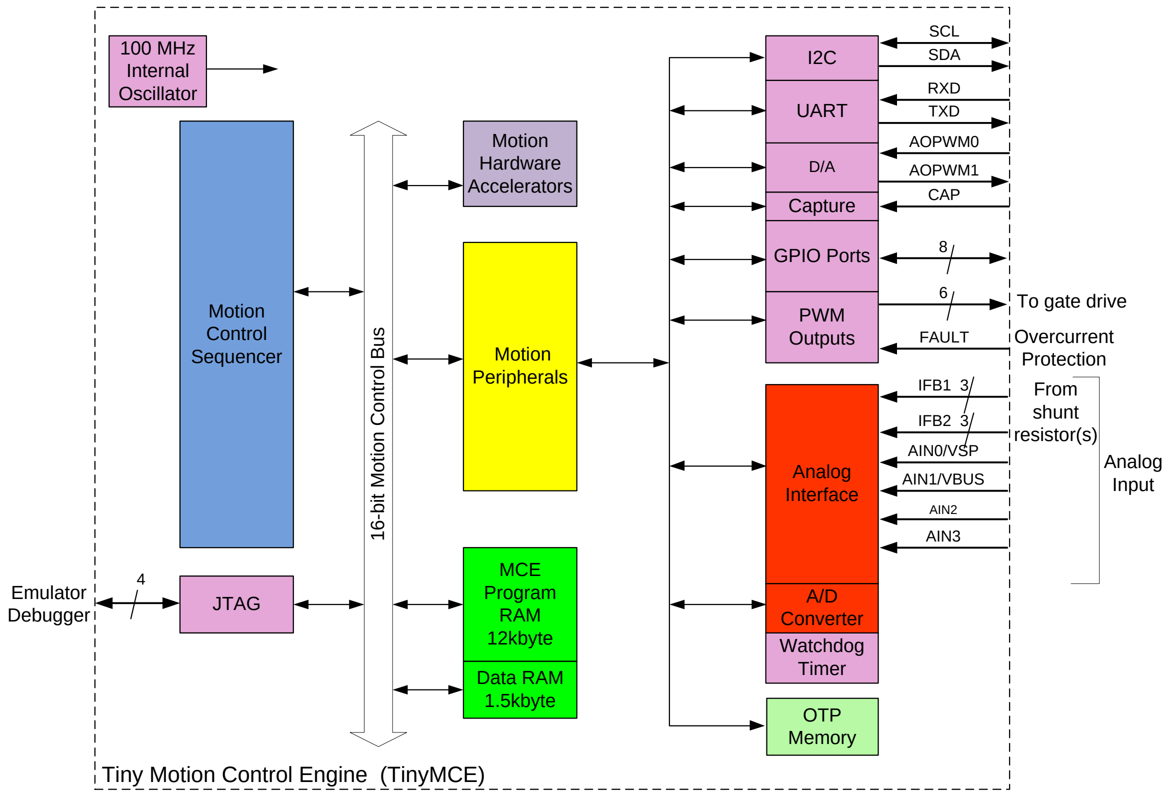


**Figure 2. IRMCK099 Internal Block Diagram**

## 1.1System Components

The TinyMCE is a 16-bit machine, with both 16-bit instructions and 16-bit wide data bus. TinyMCE can be divided into five main components, shown color-coded in **Figure 2** . The components are:

- RAM and non-volatile memory for program and data storage, shown at the center of the diagram in green
- The Motion Control Sequencer, shown at left in blue
- Hardware interface or “motion peripheral” modules, shown in yellow.
  - Analog Interface and AD converter
  - PWM Outputs
  - GPIO Ports , DA ,Capture , UART and I2C
  - Access to OTP Memory
- Motion Hardware Accelerators shown in purple support mathematical computations.
- JTAG port for debugging.

## 1.2Memory Resources

The IRMCK099 series provides three memory-mapped resources:

- Non-volatile memory for program storage
- RAM for TinyMCE program execution
- Memory-mapped motion hardware registers (MHRs)

Access to some of these resources by the TinyMCE is described in **Table 1**.

| Memory Resource | Size (bytes) | Access by TinyMCE |
|---|---|---|
| OTP Memory | 16K | Read and write |
| TinyMCE Program RAM | 12K | Read and write |
| TinyMCE Data RAM | 3K | Read and write |
| Motion Hardware Registers | 1K | Read and write |

**Table 1. Access to Memory Resources**

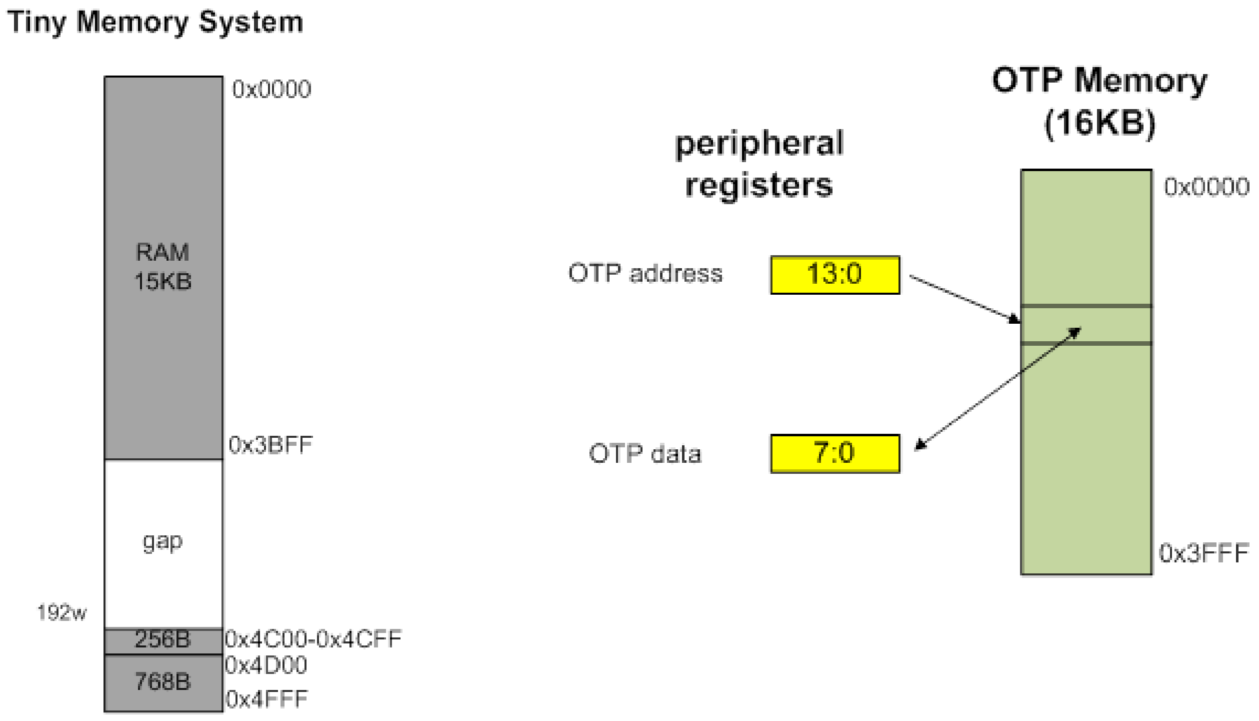


**Figure 3. Memory Map of IRMCK099 Series**

### 1.2.1 OTP Memory

For the IRMCK099, the 16K-byte OTP memory holds the TinyMCE program. The IRMCK099 hardware copies TinyMCE program from OTP to TinyMCE program RAM during the boot process and TinyMCE program executes from RAM. Refer to Section 1.4 for more information about the boot process. The firmware resides in 0x0000 to 0x2FFF address range in the OTP, Parameter block is in 0x3000 to 0x3F7F address range and factory calibration data is in 0x3F80 to 0x3FFF address range. Parameter block is used for storing multiple motor parameter or hardware configurations, There can be a total of 31 blocks each consisting of 128 bytes. These blocks can be programmed one at a time or all or few blocks at the same time using MceProgrammer. The factory calibration data consists of trimming data for the internal oscillator, band gap and also gain and offset for ADC Compensation. The breakdown of the OTP memory is shown in Figure 4.

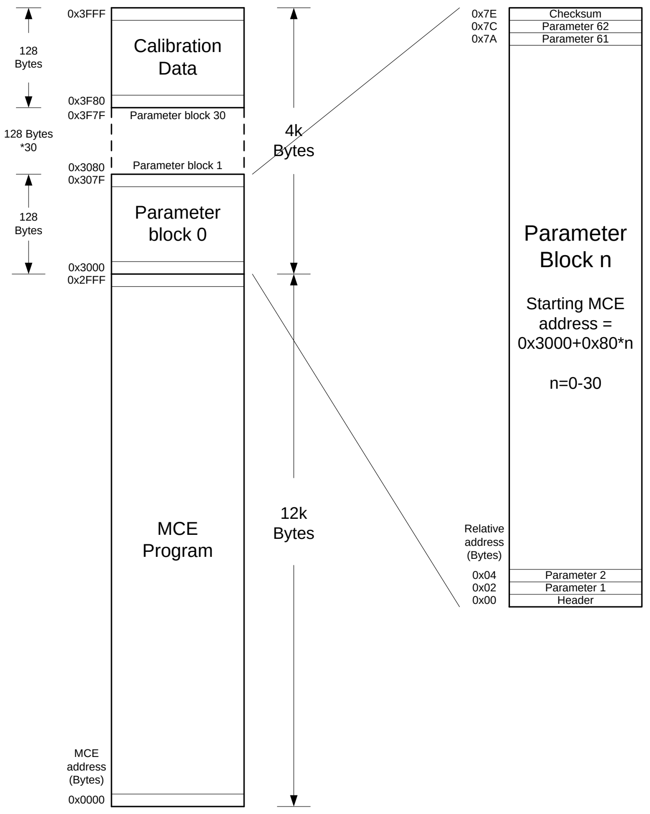


**Figure 4 .OTP Map of IRMCK099 Series**

### 1.2.2 Data RAM

TinyMCE has a Data RAM of 3K bytes. This area is used for motion firmware registers (MFRs) described in IRMCK099 Application Manual.

## 1.3Standby Mode

A standby power mode can help to increase overall system efficiency. This function will set the device in a low power consumption mode (<5mW). The 1.8V regulators are shut down, therefore the entire circuit is off except the 3.3V I/Os. Standby is customizable: user may enable the function by setting proper bit in a standby enable register.

### 1.3.1 Getting Into Standby

In normal running mode, and when the standby functionality has been enabled, the digital core monitors the voltage on AIN0 by means of the ADC. When the voltage holds lower than a programmable level for a programmable time then the TinyMCE commands the standby function by asserting STBY to high which, in turn, shuts off the 1.8V regulators.

### 1.3.2 Getting Out of Standby

The Wakeup (from Standby) function is controlled by the AIN0/WK pin. When the voltage exceeds the 0.256V, a 3.3V internal comparator is triggered and as a result the 1.8V regulators start rising. The wakeup is the same as power-up reset for the processor and since 1.8V supply is off, all RAM data is lost as a result of Standby.

### 1.3.3 Standby Control Registers

**standby_enable**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xF6A* | *Size:* | *16 bits* | *Range:* | *Boolean input 0 or 1* | *Reset value:* | *0* |

*Scaling or Notation:* 1 = Standby mode is enabled; 0 = no action.
*Description:* This register is used to enable standby mode. The standby mechanism is disabled by default. This is designed to prevent accidental system deadlock or repeated self-resetting due to incorrect or undesired configuration.

**standby_below_th**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xF6B* | *Size:* | *16 bits* | *Range:* | *Boolean input 0 or 1* | *Reset value:* | *0* |

*Scaling or Notation:* 1 = Generate TinyMCE fault condition; 0 = no action.
*Description:* This register determines the voltage level below which the 1.8V will be turned off. It drives the signal to analog circuit and a high on this signal indicates that it is a shut-down/standby command.

## 1.4Temperature Sensing

Temperature sensing block (tsense) operation is based on using an on die diode for temperature measurement.
Temperature sensing is based on measuring the voltage across the diode for two different biasing currents.
With $I_1$ and $I_2$ such currents, $V(T)_{I_x}$ voltage across the diode at temperature of T in °K with biasing current $I_x$ and $V_T$ thermal voltage of the diode:

The temperature T is proportional to the difference of the two voltage measurements:

$$T \propto \left( V(T)_{I_1} - V(T)_{I_2} \right)$$

The IRMCK099 temperature sensing is activated by setting up ADC conversions on adc_ch8.
Once the adc conversions are setup, the Firmware:

- Monitor the state machine status registers (accessible via registers in the below table) to collect the value of tsense_sub_data when ready;
- Compute the average of four tsense_sub_data values (one per current source);
- Obtain the temperature by multiplying the result for the averaged tsense_sub_data by a coefficient stored in OTP.

**Tsense_sub_data**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xF2E* | *Size:* | *16 bits* | *Range:* | *[15:0]* | *Reset value:* | *0* |

Subtraction of ADC samples = ADC(100uA)-ADC(5uA). Resulting in a Signed value. It can be read only from firmware.

## 1.5Reset Mechanism and Boot Process

A power on reset does not require any assertion of reset signal from an external circuit. The IRMCK099 series contains an analog power on reset (POR) circuit which issues reset to all internal circuits.

**Figure 5** shows the Reset Module, which contains two analog circuits, namely Under Voltage Lockout (UVCC) and Power On Reset (POR), and a 12-bit ripple counter to stretch the Reset pulse width.

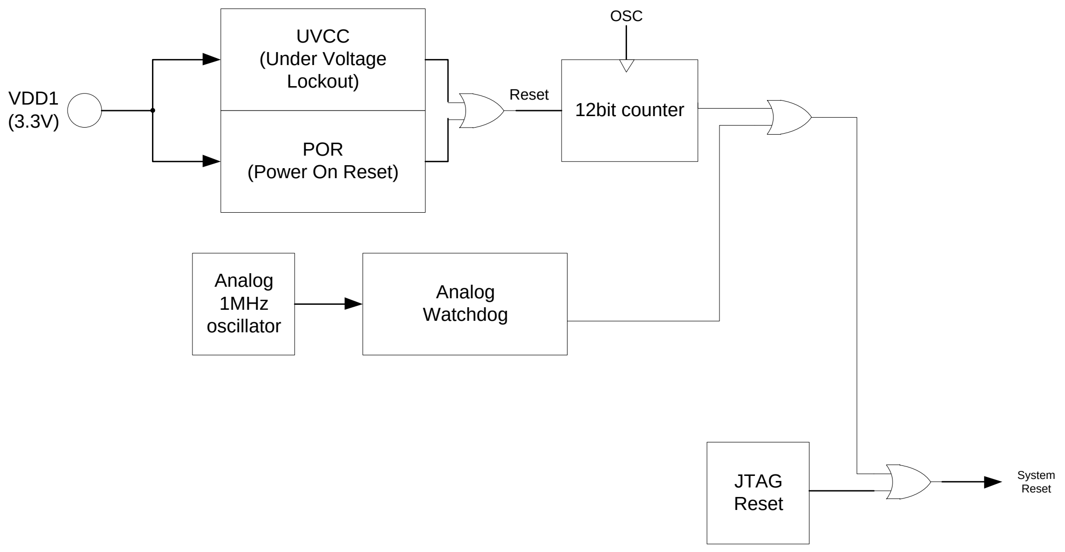


**Figure 5. Reset Module**

The reset and boot processes are closely tied together. The boot process is automatically accomplished following a proper reset sequence, which is triggered by Power On Reset. The main task of the boot process is to copy the TinyMCE program stored in internal OTP program RAM, initialize the program counter and transfer control to the sequencer. The block diagram of the reset and boot process is shown in **Figure 6**.

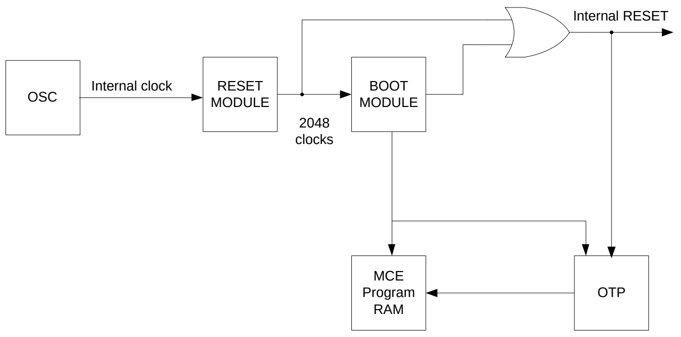


**Figure 6. Reset and Boot Process**

### 1.5.1 Boot Process

OTP bytes from 0x0000 to 0x3FFF are written to the 16-bit word at address 0x000 of the TinyMCE program RAM. This copy procedure continues until the last TinyMCE program bytes are copied from OTP offsets 0x3FFF to TinyMCE program RAM address 0x17FF. A total of 12K bytes are copied. Only after the boot sequence is done code starts running. Any additional copying from the remaining 4KBytes of OTP memory into the Data RAM can then be done by software if required. The boot process takes around 500 µSec.

## 1.6TinyMCE Processor Registers

The TinyMCE processor registers are mapped to the data RAM at addresses 0x000 – 0x013. Each register is 16 or 32 bits and appear in memory in little-endian byte order (low-order byte at the lowest address). The registers are defined as shown in **Table 2** below.

| Register | Size (bits) | TinyMCE address (hex) | Description |
|---|---|---|---|
| FLAGS | 16 | 0000 | Status bits, including TinyMCE interrupt status |
| PC | 16 | 0001 | TinyMCE program counter |
| ACC | 16 | 0002 | TinyMCE accumulator, lower 16 bits |
| ACC_EXT | 16 | 0003 | TinyMCE accumulator, upper 16 bits |
| FSR0 | 16 | 0004 | Used as a pointer for indirect reads/writes |
| FSR1 | 16 | 0005 | Used as a pointer for indirect reads/writes |
| INDR0 | 32 | 0006 | Used as access portal for indirect data (from data RAM) |
| INDR1 | 32 | 0008 | Used as access portal for indirect data (from data RAM) |
| INDR2 | 32 | 000A | Used as access portal for indirect data (from TinyMCE program RAM) |
| CTRL | 16 | 000C | Control bits, used to stop and enable the processor |
| INDR3 | 32 | 0012 | Used as access portal for indirect data (from TinyMCE program RAM) |

**Table 2. TinyMCE Processor Registers**

Most of the registers are intended for the exclusive use of the TinyMCE processor. The JTAG port may perform the following operations on these registers for debugging:

- Read and write the high-order byte of the FLAGS register, which indicates and controls TinyMCE interrupt status.
- Write to the control register to start or halt the TinyMCE processor.
- While the TinyMCE processor is halted, write to other registers to initialize their values.

The bits of the FLAGS register are defined as follows:

| Bit | Description |
|---|---|
| 0 | Zero flag for acc |
| 1 | Carry flag for acc |
| 2 | Flag bit set or cleared as a result of btst instruction |
| 3 | Zero flag for accx |
| 4 | Carry flag for accx |
| 5 | Overflow flag for acc |
| 6 | Overflow flag for accx |
| 7 | Flag bit set by ufset instruction or cleared by ufclr instruction |
| 14:8 | Reserved |
| 15 | Unsigned carry bit for accx |

**Table 3. TinyMCE Flags Register**

The bits of the CTRL register are defined as follows:

| Bit | Description |
|---|---|
| 0 | Used to control TinyMCE execution. Write “0” to halt the processor or “1” to run |
| 1 | Indicates TinyMCE status while running. “0” indicates that the processor is currently running; “1” indicates that the processor is suspended until the next SYNC pulse. |
| 2 | Used to execute a single step for debugging purposes. This is the single step “stop enable” bit. It is not the single step “launch” bit. One needs to write “1” to RUN to start the stepping. |
| 3 | Indicate TinyMCE execution status. A binary value of “00” indicates that the processor is running; a binary value of “01” indicates that the processor has been halted; a binary value of “10” indicates that the processor has been stopped because of breakpoint; a binary value of “11” indicates that the processor has been stopped because of Single-Step. |
| 5-15 | Unused |

**Table 4. TinyMCE Ctrl Register**

# 2 Motion Control Engine

The Motion Control Engine (TinyMCE) is a collection of hardware and firmware modules—the building blocks necessary to implement high efficiency sinusoidal sensorless control for Permanent Magnet motors. The full motion control algorithm: current loop and speed loop is implemented in the Firmware.

This section provides detailed descriptions of the motion peripheral registers.

The motion peripheral registers are divided into two types:

- Motion hardware registers (MHRs) are defined in hardware (RTL) and are memory mapped for access from TinyMCE. On the TinyMCE memory bus, these registers are mapped to the address range 0xE80 – 0xFFF.
- Motion firmware registers (MFRs) are defined in the TinyMCE firmware and reside in the data RAM. On the TinyMCE memory bus, these registers are in the address range 0x000 – 0x5FF.These registers are described in the Application Guide.

Motion peripheral registers (both MHRs and MFRs) range in size from 16 bits to 32 bits (although in some cases fewer than 16 bits are used). On the 16-bit TinyMCE bus, registers larger than 16 bits appear in little endian word order, low-order word first and the TinyMCE address given is for the low-order (first) word. Motion firmware registers reside in RAM and can be read and written in the same manner as ordinary RAM variables. The TinyMCE initializes data RAM to zero as one of the first tasks after the TinyMCE starts running. Reset values shown for MFRs in the following descriptions assume that TinyMCE data RAM has been initialized to zero.

Motion hardware registers are initialized by hardware at power up and reset. These registers do not reside in RAM but they are memory mapped so they can be directly read and written similarly to motion firmware registers, with the following restrictions:

- Write registers can be read and written, but reserved or undefined bits may read back as zero regardless of the value written.
- Read registers are read only, and values written to those addresses are ignored.

Certain motion firmware and motion hardware registers require initialization to a specific value, where noted in the detailed descriptions below. In these cases, the registers should be initialized before beginning execution of the TinyMCE processor. The sections below categorize the registers into functional groups and describe the purpose and format of each register. Motion hardware registers are shown with yellow shading and *it is highly recommended for the user not to access these registers.*

## 2.1Motion Hardware Registers

This section provides detailed descriptions of all the motion hardware registers referenced elsewhere in this document. The motion hardware registers are mapped to a special range of data space separate from data RAM and beginning at MCE address 0xE80.

Note that all motion hardware registers occupy either one or two words in memory, but the actual number of bits that are defined in hardware varies depending on the register. In the descriptions below, the actual hardware-defined bit size is shown. If the size is 1 – 16 bits, the register occupies one memory location; if the size is 17 – 32 bits, the register occupies two locations.

In the following sections, motion hardware registers are grouped and identified as “write registers” or “read registers”. Write registers may be read or written, but undefined bits may read back as zero regardless of the value written. Read registers are read-only and values written to those addresses are ignored.

### 2.1.1 PWM Generation and Timing Write Registers

The following write registers are used to set up PWM generation and timing. The MCE processor is customized for motor control and PWM cycle timing is central to its operation. Task scheduling and ADC sampling are based on PWM cycle timing. The following terminology is used:

**Master PWM Cycle**

The master PWM cycle or master PWM period refers to the duration of the basic (shortest, highest-frequency) PWM cycle time. When the system supports only a motor, the master PWM cycle is set to the motor PWM frequency.

**PWM Cycle Period**

The master PWM frequency defines the basic time period in the system. All other scheduled events are based on the master PWM cycle time or a multiple of it. These events include task scheduling (timewheel), level changes for PWM output signals, ADC sample-and-hold times and MCE pin timers.

#### 2.1.1.1 Configuring Master PWM Cycle Timing

The master PWM frequency is configured using the master_duration register, which specifies the number of system clock cycles in one complete master PWM cycle. In addition, the master_time_incr register must be configured with an increment value for the 24-bit hardware “phase” counter, which is increased by the specified value on each clock cycle such that it counts from 0 to 0xFFFFFF in one complete master PWM cycle.

**master_duration**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xE82* | *Size:* | *16 bits* | *Range:* | *Unsigned input 0 – 65,535* | *Reset value:* | *0* |

*Scaling or Notation:* See description.

*Description:* This register defines the length of a master PWM cycle, specified as a number of system clock cycles. For example, if the system clock frequency is 100 MHz, a value of 20,000 in master_duration configures the master PWM frequency to 5 KHz.

**master_time_incr**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xE84* | *Size:* | *24 bits* | *Range:* | *Unsigned input 0 – 0xFFFFFF* | *Reset value:* | *0* |

*Scaling or Notation:* See description.

*Description:* This register defines the increment value for the hardware "phase" counter, which is updated on every clock cycle. The value should be calculated so that the phase counter reaches 0xFFFFFF at the end of the master cycle. That is, master_time_incr = 0x1000000 / master_duration.

#### 2.1.1.2 Configuring PWM Cycle Periods

Each period register determines the number of master cycles in the associated PWM cycle period. The period register is set to numMasterCycles – 1, where numMasterCycles is the number of master cycles in the period. The value of each 4-bit period register can have the range 0 – 15. One or more 20-bit timers are associated with each period register and the period determines the maximum count for the associated timers. Specifically, the maximum count value is 0x*p*FFFF, where *p* = the value of the associated period register.

**Table 5** summarizes the period registers. The period registers associated with PWM output signals and ADC sample timing are fully described below in this document.

| **Register** | **Description** |
|---|---|
| period_1 | Period for motor PWM cycle timing and sample timing for ADC channels 0 – 3. |
| period_3 | Sample timing for ADC channel 8. |
| period_ad9 | Period for ADC channel 9 sample timing. |
| period_ad10 | Period for ADC channel 10 sample timing. |
| period_ad11 | Period for ADC channel 11 sample timing. |

**Table 5. Summary of Period Registers**

**period_1**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xEA4* | *Size:* | *4 bits* | *Range:* | *Unsigned input 0 – 15* | *Reset value:* | *0* |

| | |
|---|---|
| *Scaling or Notation:* | See description. |
| *Description:* | This register determines the number of master cycles in each motor PWM cycle. It should be set to numMasterMotor – 1, where numMasterMotor is the number of master cycles in each motor PWM cycle. |

This register determines the valid range for the following timer registers:

| | |
|---|---|
| ton_0_1 | Motor U phase turn on time |
| toff_0_1 | Motor U phase turn off time |
| ton_1_1 | Motor V phase turn on time |
| toff_1_1 | Motor V phase turn off time |
| ton_2_1 | Motor W phase turn on time |
| toff_2_1 | Motor W phase turn off time |
| tmr_a_1 | Sample-and-hold time for ADC channel 0 |
| tmr_b_1 | Sample-and-hold time for ADC channel 1 |
| tmr_c_1 | Sample-and-hold time for ADC channel 2 |
| tmr_d_1 | Sample-and-hold time for ADC channel 3 |

**period_3**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xEA6* | *Size:* | *4 bits* | *Range:* | *Unsigned input 0 – 15* | *Reset value:* | *0* |

**period_ad9**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xEA7* | *Size:* | *4 bits* | *Range:* | *Unsigned input 0 – 15* | *Reset value:* | *0* |

**period_ad10**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xEA8* | *Size:* | *4 bits* | *Range:* | *Unsigned input 0 – 15* | *Reset value:* | *0* |

**period_ad11**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xEA9* | *Size:* | *4 bits* | *Range:* | *Unsigned input 0 – 15* | *Reset value:* | *0* |

*Scaling or Notation:* See description.

*Description:* These registers determine the number of master cycles in each ADC sampling period for ADC channels 9, 10 and 11. Each register should be set to numMasterADC – 1, where numMasterADC is the desired number of master cycles in the ADC sampling period. These registers are used to configure the sample-and-hold times for the ADC channels. The 20-bit value of each register determines the point within the PWM cycle at which the sample-and-hold operation is requested for the associated ADC channel. The valid range for each register depends on an associated “period” register, which configures the duration of the PWM cycle in relation to a “master” PWM cycle. Table 6 shows the correspondence between these registers, ADC channels, and associated period registers.

| ADC Channel | Sample Time Register | Associated Period Register |
|---|---|---|
| CH 0 | tmr_a_1 | period_1 |
| CH 1 | tmr_b_1 | period_1 |
| CH 2 | tmr_c_1 | period_1 |
| CH 3 | tmr_d_1 | period_1 |
| CH 8 | tmr_3 | period_3 |
| CH 9 | tmr_ad9 | period_ad9 |
| CH 10 | tmr_ad10 | period_ad10 |
| CH 11 | tmr_ad11 | period_ad11 |

Table 6. ADC Sample-and-Hold Time Registers

For each register, the 20-bit value determines the point within the associated PWM cycle period at which the sample-and-hold operation is requested. The valid range of timer values depends on the value of the associated period register. If the period register is set to a value *p*, then the timer count value can range from 0x00000 to 0xpFFFF.

**deadtime_en**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xE86* | *Size:* | *3 bits* | *Range:* | *Unsigned input with bit field definitions* | *Reset value:* | *0* |

*Scaling or Notation:* 1 = Enable deadtime insertion

0 = Disable deadtime insertion

*Description:* This register enables and disables deadtime insertion for PWM outputs.

| | |
|---|---|
| Bit 0 | **Motor Deadtime Enable** |
| Bit 1 | *Reserved. Should be set to 0.* |
| Bit 2 | *Reserved. Should be set to 0.* |

**deadtime (PwmDeadTm)**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xE87* | *Size:* | *10 bits* | *Range:* | *Unsigned input 0 – 1023* | *Reset value:* | *0* |

*Scaling or Notation:* 1 = one system clock cycle.

*Description:* Inverter blanking time for avoiding shoot through between high side and low side switching devices.

**active_pol (ActivePol)**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xE88* | *Size:* | *6 bits* | *Range:* | *Unsigned input with bit field definitions* | *Reset value:* | *0* |

*Scaling or Notation:* See description.

*Description:* This register controls active polarity of PWM outputs as shown below.

| | | |
|---|---|---|
| Bit 0 | **Motor GateSenLow** | |
| | 0 | active high for low side switches |
| | 1 | active low for low side switches |
| Bits 1 – 2 | *Reserved. Should be set to 0.* | |
| Bit 3 | **Motor GateSenHigh** | |
| | 0 | active high for high side switches |
| | 1 | active low for high side switches |
| Bit 4 -5 | *Reserved. Should be set to 0.* | |

**pwm_inv_1**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xEAA* | *Size:* | *3 bits* | *Range:* | *Unsigned input with bit field definitions* | *Reset value:* | *undefined* |

*Scaling or Notation:* 1 = invert signal

0 = do not invert

*Description:* This register determines whether or not motor PWM output signals are inverted prior to deadtime insertion. The bits correspond to PWM output signals as follows:

| | | |
|---|---|---|
| Bit 0 | **PWMU** | U phase |
| Bit 1 | **PWMV** | V phase |
| Bit 2 | **PWMW** | W phase |

**pin_enable**

*Address:* *0xE89* *Size:* *6 bits* *Range:* *Unsigned input with bit field definitions* *Reset value:* *0*

*Scaling or Notation:* See description.

*Description:* This register enables and disables PWM outputs.

| | | |
|---|---|---|
| Bit 0 | **Motor Enable Low** | |
| | 0 | disable low side switches |
| | 1 | enable low side switches |
| Bits 1 – 2 | *Reserved. Should be set to 0.* | |
| Bit 3 | **Motor Enable High** | |
| | 0 | disable high side switches |
| | 1 | enable high side switches |
| Bit 4 -5 | *Reserved. Should be set to 0.* | |

**port_ctrl0**

*Address:* *0xE80* *Size:* *14 bits* *Range:* *Unsigned input with bit field definitions* *Reset value:* *0*

*Scaling or Notation:* See description.

*Description:* This register is used to set the level of each PWM output. For each two-bit field defined below, the following values apply:

| | |
|---|---|
| 00 | Tristate |
| 01 | PWM timer output (normal operation) |
| 10 | Set low |
| 11 | Set high |

| | |
|---|---|
| Bits 0 – 1 | UH |
| Bits 2 – 3 | UL |
| Bits 4 – 5 | VH |

| | |
|---|---|
| Bits 6 – 7 | VL |
| Bits 8 – 9 | WH |
| Bits 10 – 11 | WLUL |
| Bits 12 – 13 | *Reserved. Should be set to 0.* |

**ton_0_1**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xEAC* | *Size:* | *20 bits* | *Range:* | *Unsigned input 0 – 1,048,575* | *Reset value:* | *0* |

**toff_0_1**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xEAE* | *Size:* | *20 bits* | *Range:* | *Unsigned input 0 – 1,048,575* | *Reset value:* | *0* |

**ton_1_1**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xEB0* | *Size:* | *20 bits* | *Range:* | *Unsigned input 0 – 1,048,575* | *Reset value:* | *0* |

**toff_1_1**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xEB2* | *Size:* | *20 bits* | *Range:* | *Unsigned input 0 – 1,048,575* | *Reset value:* | *0* |

**ton_2_1**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xEB4* | *Size:* | *20 bits* | *Range:* | *Unsigned input 0 – 1,048,575* | *Reset value:* | *0* |

**toff_2_1**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xEB6* | *Size:* | *20 bits* | *Range:* | *Unsigned input 0 – 1,048,575* | *Reset value:* | *0* |

*Scaling or Notation:* 1 = 1/65535 of master PWM cycle.

| | |
|---|---|
| *Description:* | These registers are used to set the turn-on and turn-off times for the motor PWM outputs. Each register configures the turn-on time or turn-off time for one of the motor PWM output signals, as follows: |

| | |
|---|---|
| ton_0_1 | Motor U phase turn on time |
| toff_0_1 | Motor U phase turn off time |
| ton_1_1 | Motor V phase turn on time |
| toff_1_1 | Motor V phase turn off time |
| ton_2_1 | Motor W phase turn on time |
| toff_2_1 | Motor W phase turn off time |

For each register, the 20-bit value determines the point within the motor PWM cycle at which the level of the output signal is changed. The valid range of values depends on the motor PWM period configuration in register period_1. If the period_1 register is set to a value *p*, then the ton/toff count value can range from 0x00000 to 0x*p*FFFF. For a signal level change at a particular point within the motor PWM cycle, set the count value to the corresponding percentage of the maximum value.

Example: period_1 is 0 and the maximum count value is 0x0FFFF. Set ton_0_1 = 0x04000 to turn on the motor U phase signal after the first quarter of the motor PWM cycle and set toff_0_1= 0xC000 to turn off the motor U phase signal after three quarters of the cycle.

### 2.1.2 PWM Generation and Timing Read Registers

The following read registers provide information regarding PWM signal status.

**pwm_lines (pwm_lines)**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xF1A* | *Size:* | *13 bits* | *Range:* | *Unsigned output with bit field definitions* | *Reset value:* | *undefined* |

| | |
|---|---|
| *Scaling or Notation:* | See description. |
| *Description:* | This register provides internal PWM gating bits. |

| | | |
|---|---|---|
| Bits 0 – 5 | Unused | |
| Bit 6 | **PWMUH** | U phase high side output |
| Bit 7 | **PWMUL** | U phase low side output |
| Bit 8 | **PWMVH** | V phase high side output |
| Bit 9 | **PWMVL** | V phase low side output |
| Bit 10 | **PWMWH** | W phase high side output |
| Bit 11 | **PWMWL** | W phase low side output |
| Bit 12 | *Reserved.* | |

### 2.1.3 A/D Converter Write Registers

The following write registers configure A/D sampling parameters.

**adc_ch0**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xE94* | *Size:* | *4 bits* | *Range:* | *Unsigned input 1 – 14* | *Reset value:* | *0* |

**adc_ch1**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xE95* | *Size:* | *4 bits* | *Range:* | *Unsigned input 1 – 14* | *Reset value:* | *0* |

**adc_ch2**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xE96* | *Size:* | *4 bits* | *Range:* | *Unsigned input 1 – 14* | *Reset value:* | *0* |

**adc_ch3**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xE97* | *Size:* | *4 bits* | *Range:* | *Unsigned input 1 – 14* | *Reset value:* | *0* |

**adc_ch8**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xE9C* | *Size:* | *4 bits* | *Range:* | *Unsigned input 1 – 14* | *Reset value:* | *0* |

**adc_ch9**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xE9D* | *Size:* | *4 bits* | *Range:* | *Unsigned input 1 – 14* | *Reset value:* | *0* |

**adc_ch10**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xE9E* | *Size:* | *4 bits* | *Range:* | *Unsigned input 1 – 14* | *Reset value:* | *0* |

**adc_ch11**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xE9F* | *Size:* | *4 bits* | *Range:* | *Unsigned input 1 – 14* | *Reset value:* | *0* |

*Scaling or Notation:* See description.

*Description:* These registers are used to select an ADC input to sample on the associated ADC channel. Table 6 shows which ADC channel is associated with each of the above registers. There are a total of 14 valid inputs, 1 – 14. The values 0 and 15 are unused. (If the value 0 or 15 is specified, input 14 is selected.)

The same input number may be configured for more than one ADC channel. (That is, more than one of the above registers may be configured with the same value.)

**tmr_a_1**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xEB8* | *Size:* | *20 bits* | *Range:* | *Unsigned input 0 – 1,048,575* | *Reset value:* | *0* |

**tmr_b_1**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xEBA* | *Size:* | *20 bits* | *Range:* | *Unsigned input 0 – 1,048,575* | *Reset value:* | *0* |

**tmr_c_1**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xEBC* | *Size:* | *20 bits* | *Range:* | *Unsigned input 0 – 1,048,575* | *Reset value:* | *0* |

**tmr_d_1**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xEBE* | *Size:* | *20 bits* | *Range:* | *Unsigned input 0 – 1,048,575* | *Reset value:* | *0* |

**tmr_3**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xEDA* | *Size:* | *20 bits* | *Range:* | *Unsigned input 0 – 1,048,575* | *Reset value:* | *0* |

**tmr_ad9**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xEDC* | *Size:* | *20 bits* | *Range:* | *Unsigned input 0 – 1,048,575* | *Reset value:* | *0* |

**tmr_ad10**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xEDE* | *Size:* | *20 bits* | *Range:* | *Unsigned input 0 – 1,048,575* | *Reset value:* | *0* |

**tmr_ad11**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xEE0* | *Size:* | *20 bits* | *Range:* | *Unsigned input 0 – 1,048,575* | *Reset value:* | *0* |

*Scaling or Notation:* 1 = 1/65535 of master PWM cycle.

*Description:* These registers are used to configure the sample-and-hold times for the ADC channels. The 20-bit value of each register determines the point within the PWM cycle at which the sample-and-hold operation is requested for the associated ADC channel. The valid range for each register depends on an associated “period” register, which configures the duration of the PWM cycle in relation to a “master” PWM cycle. shows the correspondence between these registers, ADC channels, and associated period registers.

For each register, the 20-bit value determines the point within the associated PWM cycle period at which the sample-and-hold operation is requested. The valid range of timer values depends on the value of the associated period register. (The period register configures a cycle time as a multiple of the master PWM cycle). If the period register is set to a value *p*, then the timer count value can range from 0x00000 to 0xpFFFF. For example, if period is 2 the count can range from 0x00000 to 2FFFF. For a sample-and-hold request at a particular point within the cycle, set the count value to the corresponding percentage of the maximum value. Example: For tmr_a_1, the maximum count value is 0x0FFFF since period_1 is 0. (The motor PWM period is the master and global PWM cycle.) Set tmr_a_1 = 0x04000 to request sample and hold after the first quarter of the motor PWM cycle.

**adc_base_addr**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xEA0* | *Size:* | *7 bits* | *Range:* | *Unsigned input 0 – 0x7F* | *Reset value:* | *7* |

*Scaling or Notation:* Address specified as a multiple of 32 words.

*Description:* This register sets the base address of an area in data RAM where hardware stores the ADC data. The address is specified as a multiple of 32 words, for an address range of 0x000 – 0xFE0. For example, the value 0x08 sets the base address of the ADC data area to 0x100.

**adc_sample_ena**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xEA1* | *Size:* | *12 bits* | *Range:* | *Unsigned input with bit field definitions* | *Reset value:* | *0* |

*Scaling or Notation:* 1 = enable

0 = disable

*Description:* This register enables and disables individual ADC channels. Bits 0 – 11 represent ADC channels 0 – 11, respectively, and each bit can be set to 1 to enable the associated channel or 0 to disable it. For example, the value 0x00F enables channels 0 – 3 and disables channels 4 – 11.

**adc_clkdiv**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xEA3* | *Size:* | *6 bits* | *Range:* | *Unsigned input with bit field definitions* | *Reset value:* | *0* |

*Scaling or Notation:* See description.

*Description:* This register defines the frequency at which the ADC operates. The value of adc_clkdiv divides the system clock to create the ADC clock, as follows:

ADC clock frequency = SYSCLK / (adc_clkdiv + 1)

When the value of adc_clkdiv is zero, the ADC is not enabled.

**priority_low**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xF06* | *Size:* | *32 bits* | *Range:* | *Unsigned input with bit field definitions* | *Reset value:* | *0xFEDCBA95* |

**priority_high**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xF08* | *Size:* | *16 bits* | *Range:* | *Unsigned input with bit field definitions* | *Reset value:* | *0x4321* |

*Scaling or Notation:* See description.

*Description:* These registers define the priority of each ADC channel. When multiple channels request conversion at the same time, the requesting channel with the higher priority is serviced first. If two requesting channels are configured with the same priority, the higher-numbered channel is serviced first. A four-bit specifying a priority value (0 – 15) is assigned to each of the ADC channels as shown below. A value of 0 gives the lowest priority and 15 gives the highest. Multiple channels may be configured at the same priority.

**priority_low**

| | |
|---|---|
| Bits 0 – 3 | Channel 0 |
| Bits 4 – 7 | Channel 1 |
| Bits 8 – 11 | Channel 2 |
| Bits 12 – 15 | Channel 3 |
| Bits 15 - 31 | Reserved |

**priority_high**

| | |
|---|---|
| Bits 0 – 3 | Channel 8 |
| Bits 4 – 7 | Channel 9 |
| Bits 8 – 11 | Channel 10 |
| Bits 12 – 15 | Channel 11 |

### 2.1.4 A/D Converter Read Registers

The following read registers provide information regarding A/D converter sampling status.

**adc_bank**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xF19* | *Size:* | *6 bits* | *Range:* | *Unsigned output with bit field definitions* | *Reset value:* | *0* |

*Scaling or Notation:* 0 = Current bank is 0

1 = Current bank is 1

*Description:* Each bit in this register gives the current ADC fill bank for a section of the ADC data, as follows:

| | |
|---|---|
| Bit 0 | Current bank for ADC channels 0 – 3, associated with period_1 |
| Bit 1 | Reserved |
| Bit 2 | Current bank for ADC channel 8, associated with period_3 |
| Bit 3 | Current bank for ADC channel 9, associated with period_ad9 |
| Bit 4 | Current bank for ADC channel 10, associated with period_ad10 |
| Bit 5 | Current bank for ADC channel 11, associated with period_ad11 |

The term "current bank" refers to the bank that contains the most recent valid data samples. The bank that is not current should not be accessed since those locations are waiting to be filled or in the process of being filled.

Hardware automatically switches the current bank at the beginning of the PWM cycle period, as defined by the period register associated with the ADC channel group. During any given PWM cycle period, the current bank is the one that was filled during the previous PWM cycle period. For example, if adc_bank bit 0 = 1, then bank 1 contains valid data for ADC channels 0 - 3 sampled in the previous PWM cycle period and bank 0 will be filled with new data during the current PWM cycle period.

**sample_req**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xF1B* | *Size:* | *12 bits* | *Range:* | *Unsigned output with bit field definitions* | *Reset value:* | *0* |

*Scaling or Notation:* 0 = Channel is not enabled or has been serviced

| | |
|---|---|
| | 1 = Channel is enabled and service is pending |
| *Description:* | Each bit in this register gives the current sampling status for an ADC channel, with bits 0 – 11 corresponding to ADC channels 0 – 11, respectively. |

### 2.1.5 Timewheel Write Registers

The following write registers are associated with timewheel setup.

**exe_active**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xEE2* | *Size:* | *1 bit* | *Range:* | *Boolean input*<br>*0 or 1* | *Reset value:* | *0* |

| | |
|---|---|
| *Scaling or Notation:* | 0 = disabled.<br>1 = enabled. |

*Description:* This register enables and disables timewheel operation. When the timewheel is disabled, hardware does not begin execution of any timewheel functions regardless of the MCE processor state. When the timewheel is enabled, PWM cycle timing triggers task scheduling as specified in the timewheel table.

**timewheel_addr**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xEE4* | *Size:* | *12 bits* | *Range:* | *Unsigned input*<br>*Data RAM address* | *Reset value:* | *0* |

| | |
|---|---|
| *Scaling or Notation:* | See description. |
| *Description:* | This register holds the address of the timewheel table in data RAM. |

### 2.1.6 Timewheel Read Registers

The following read registers are associated with timewheel operation. The timewheel table setup determines the global PWM cycle as defined above. PWM output signals, ADC sampling and MCE pin timers are scheduled according to configured PWM cycle periods. A PWM cycle period is a multiple of the master PWM cycle time and cannot exceed the global PWM cycle time. Period registers are used to define the PWM cycle period for various operations as summarized in Table 5.

**exe_prog_count**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xF20* | *Size:* | *12 bits* | *Range:* | *Unsigned output*<br>*0 - 4095* | *Reset value:* | *0* |

| | |
|---|---|
| *Scaling or Notation:* | See description. |
| *Description:* | This register holds the value of a counter that is incremented by one each time hardware executes an entry in the timewheel table. After execution has wrapped back to the start of the timewheel table, hardware always clears the counter to zero when the second timewheel table entry is executed.<br><br>Note that the counter begins at zero after reset, but is not cleared to zero again until after the first wrap back to the start of the table. That is, the very first time the second timewheel table entry is executed (before the first wrap), the counter is *not* cleared to zero. |

**exe_pwm_sync**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xF21* | *Size:* | *2 bit* | *Range:* | *Boolean output* | *Reset value:* | *0* |
| | | | | | *0 or 1* | | |

*Scaling or Notation:* 0 = Not executing second timewheel table entry.

1 = Currently executing second timewheel table entry.

*Description:* When PWM timing is configured with one master cycle for each global cycle (motor only, no PFC), the exe_pwm_sync register is set to 1 after the second time hardware wraps back to the start of the timewheel table and remains = 1 thereafter.

When PWM timing is configured with multiple master cycles for each global cycle (motor and PFC), the exe_pwm_sync register is set to 1 during execution of the second timewheel table entry and is 0 otherwise.

### 2.1.7 Factory Calibration Registers

**Factory_Calib_checksum_lo**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0x3FE0* | *Size:* | *8 bits* | *Range:* | *Unsigned output* | *Reset value:* | *NA* |
| | | | | | *0 - 255* | | |

*Scaling or Notation:* See description.

*Description:* Accuracy of the factory calibration registers (0x3FE0-0x3FFF) is verified via checksum. Factory_Calib_checksum_lo holds the least significant byte of the checksum value of factory calibration registers.

**Factory_Calib_checksum_hi**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0x3FE1* | *Size:* | *8 bits* | *Range:* | *Unsigned output* | *Reset value:* | *NA* |
| | | | | | *0 - 255* | | |

*Scaling or Notation:* See description.

*Description:* Accuracy of the factory calibration registers (0x3FE0-0x3FFF) is verified via checksum. Factory_Calib_checksum_hi holds the most significant byte of the checksum value of factory calibration registers.

**Reserved**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0x3FF0-0x3FF3* | *Size:* | *8 bits* | *Range:* | *Unsigned output* | *Reset value:* | *NA* |
| | | | | | *0 - 255* | | |

*Scaling or Notation:* See description.

*Description:* These registers are reserved for future use as factory calibration registers.

**AdGainx1024**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0x0D3(MCE)* | *Size:* | *16 bits* | *Range:* | *Unsigned output* | *Reset value:* | *NA* |
| | *0x3FF4(OTP)* | | | | *-32768 to 32767* | *1024* | |

*Scaling or Notation:* See description.

*Description:* On-chip ADC output is precisely calibrated for each IC by configuring AdGainx1024 and AdOfset values in the OTP during factory calibration. The trim value for this register is stored in the OTP in little endian order. During the boot up, this value is transferred to the RAM register from its corresponding OTP location. However, if the value is not stored in OTP then *'Reset Value'* of this register is transferred to the RAM register by default.

**AdOfst**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0x0D2(MCE)* | *Size:* | *16 bits* | *Range:* | *Unsigned output* | *Reset value:* | *NA* |
| | *0x3FF6(OTP)* | | | | *0 to 65535* | *0* | |

*Scaling or Notation:* See description.

*Description:* On-chip ADC output is precisely calibrated for each IC by configuring AdGainx1024 and AdOfset values in the OTP during factory calibration. The trim value for this register is stored in the OTP in little endian order. During the boot up, this value is transferred to the RAM register from its corresponding OTP location. However, if the value is not stored in OTP then *'Reset Value'* of this register is transferred to the RAM register by default.

**Tsense_25c**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0x0FB(MCE)* | *Size:* | *16 bits* | *Range:* | *Unsigned output* | *Reset value:* | *NA* |
| | *0x3FFA(OTP)* | | | | *0 to 65535* | *1012* | |

*Scaling or Notation:* See description.

*Description:* On-chip temperature sense output is precisely calibrated for each IC by configuring Tsense_25c in the OTP during factory calibration. Tsense_25c register holds the 16 bit unsigned value of the Temperature Sense Offset; obtained after factory calibration of AdGainx1024. The trim value for this register is stored in the OTP in little endian order. During the boot up, this value is transferred to the RAM register from its corresponding OTP location. However, if the value is not stored in OTP then *'Reset Value'* of this register is transferred to the RAM register by default.

**Trimbandgap**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0xF54(MCE)* | *Size:* | *16 bits* | *Range:* | *Unsigned output* | *Reset value:* | *NA* |
| | *0x3FFD(OTP)* | | | | *0 to 65535* | *172* | |

*Scaling or Notation:* See description.

*Description:* Accuracy of on-chip 1.2V and 1.8V voltage references is calibrated by trimming the bandgap reference value in the OTP during factor calibration. Trimpbandgap register holds the 16 bit unsigned value of the bandgap reference. The trim value for this register is stored in the OTP in little endian order. During the boot up, this value is transferred to the RAM register from its corresponding OTP location. However, if the value is not stored in OTP then *'Reset Value'* of this register is transferred to the RAM register by default.

**Trimcurrentsrc**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0x0F55(MCE)* | *Size:* | *8 bits* | *Range:* | *Unsigned output* | *Reset value:* | *NA* |
| | *0x3FFE(OTP)* | | | | *0 to 255* | *128* | |

*Scaling or Notation:* See description.

*Description:* An on-chip precision current source drives the bandgap reference as well as the on-chip oscillator. Any inaccuracy in this current source is propagated to the bandgap reference voltage as well as oscillator frequency. This current source can be precisely trimmed by configuring Trimcurrentsrc register value in the OTP during factory calibration. The trim value for this register is stored in the OTP. During the boot up, this value is transferred to the RAM register from its corresponding OTP location. However, if the value is not stored in OTP then *'Reset Value'* of this register is transferred to the RAM register by default.

**Trimosc**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *Address:* | *0x0F56(MCE)* | *Size:* | *8 bits* | *Range:* | *Unsigned output* | *Reset value:* | *NA* |
| | *0x3FFF(OTP)* | | | | *0 to 255* | *102* | |

| | |
|---|---|
| *Scaling or Notation:* | See description. |
| *Description:* | Accuracy of on-chip oscillator is calibrated by trimming the oscillator frequency precisely to 100MHz by configuring Trimosc register value in the OTP during factory calibration. Trimosc register holds the trim values for the on-chip oscillator. The trim value for this register is stored in the OTP. During the boot up, this value is transferred to the RAM register from its corresponding OTP location. However, if the value is not stored in OTP then *'Reset Value'* of this register is transferred to the RAM register by default. |

# 3 Programming the Control IC

This section describes how to load the MCE program to RAM for testing and how to program the MCE programs to non-volatile memory. Several options for programming are described below:

1. Use the IRMCx Download Cable (IRCable) to program the IC.
2. Use on-board programming with one of the IRMCSxxxx reference design boards through the provided debug connector.
3. Use customer-designed hardware (final application board) through the JTAG or UART interface using the IRMCS-ISO V3 isolation board.

OTP programming is described in Section 3.1 and in Section 3.2 usage of the MCEProgrammer is described followed by the instructions for using IRCable in Section 3.3. Section 3.4 describes the RAM download procedure and custom programming methods are covered in Section 3.5.

As input, MCEProgrammer imports a design's MCE program (.bin file) converts the program files into a memory image in the proper format for storage in OTP memory and then programs the memory image to OTP.

## 3.1Programming OTP Memory

This section describes how to program OTP memory on the IRMCK099 ICs via IRCable.

OTP memory is programmed using the MCEProgrammer utility, which can perform direct OTP programming by communicating with the IRMCx Download Cable (IRCable).

As input, MCEProgrammer imports a design's MCE program (.bin file) converts the program files into a memory image in the proper format for storage in OTP memory and then programs the memory image to OTP.

To program the OTP, the user has several options:

1. Use the reference design board with IRCable to program the IC on-board through the provided debug connector. This option is described in Section 3.3.
2. Design custom hardware to communicate with the IRMCK099 series IC through the JTAG pins. This option is described in Section 3.5.

As described in section 1.2.1, OTP Memory is divided into 3 sections i.e. Calibration Data, Parameter Blocks and MCE Program. This section describes the Parameter Block section of the OTP and presents a step by step procedure to create Parameter Block File.

### 3.1.1 Multiple Parameter Blocks

There are totally 31 parameter blocks that can be used to store control parameters. Each parameter block is 128 bytes in size. Multiple parameter blocks can be programmed in order to support different motor types. Each parameter block has a field called “MotorID” which is to identify the motor type that it supports.

The parameter block that is programmed into the OTP is the output of TinyWizard saved into .txt file. Every time there is parameter change, new block can be programmed into the unused parameter block using MCEProgrammer. MCEProgrammer will search if there is any block that has already been programmed in OTP memory and if there was no block that was programmed before, the values will be written to block 1 located at 0x3F7F. If there was already a block programmed, the next available block will be programmed. For example, if there were 3 blocks already programmed, then block 4 will be programmed at 0x3D80.

The starting address of each block can be calculated as shown below:

Starting Block Address in OTP = 0x3000 + 0x80 x (31 –n)   where n =1 to 31

For example, parameter block 2 starting MCE address =0x3000 + 0x80 x (0x1D) = 0x3E80

Bit [10:9] in register HwConfig is used to enable/disable the multiple motor support and also configure the source for MotorID selection. The HwConfig value is always used from the latest block or the last block written to the OTP.

Multiple motor support is disabled if HwConfig[10:9]=00, MCE firmware will search for the most recent valid parameter block and use it to configure the registers.

Multiple motor support is enabled if HwConfig[10:9]=01, 10, or 11:

- HwConfig[10:9]=01, MotorID will be read from AIN3 input
- HwConfig[10:9]=10, MotorID will be read from AIN0 input
- HwConfig[10:9]=11, MotorID will be selected by writing a value between 0 to 255 to register MotorIDSel.

If multiple motor support is enabled in the most recent block and MotorID is set (by AIN3/AIN0 or UART), it will start to search for parameter block which has matching MotorID and use it to configure the parameters.

Once the 31 parameter blocks are used up, IC cannot be programmed with new parameter. Due to limited number of parameter block, during development, it’s better to tune the motor in RAM and program the parameter block only when motor tuning is done. Figure 4 shows the memory address for the parameter block and in Figure 7 the OTP parameter load procedure is shown in a flow chart.

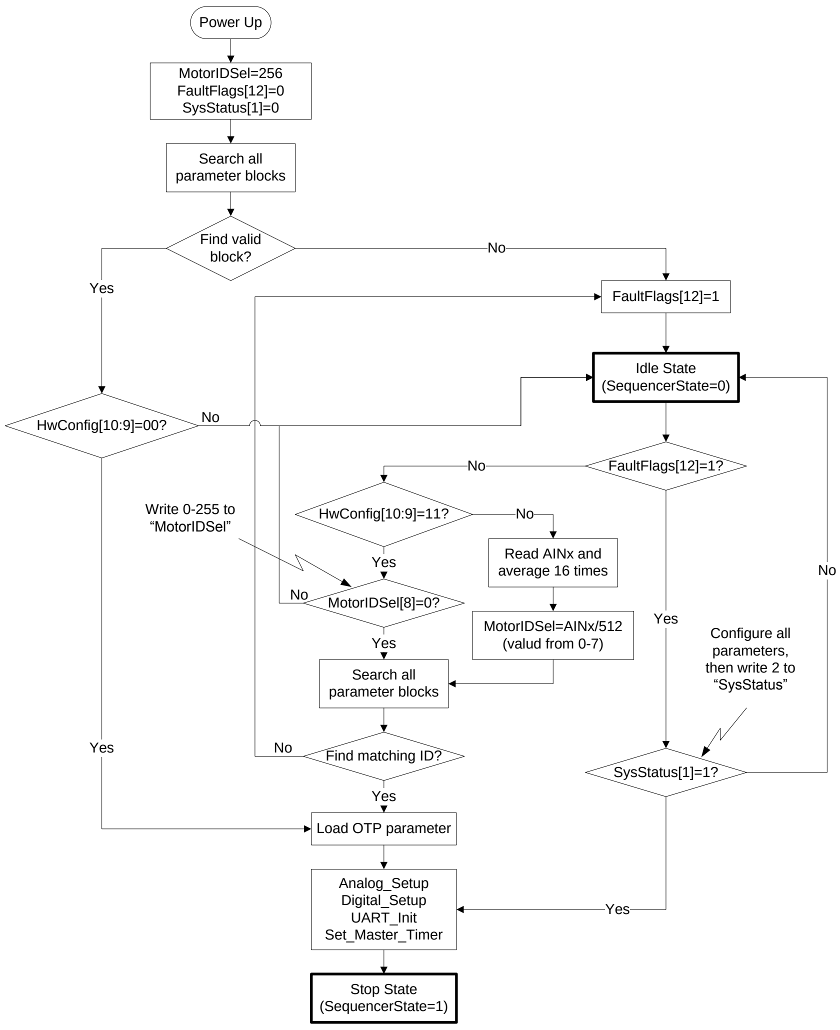


Figure 7: OTP parameter load procedure

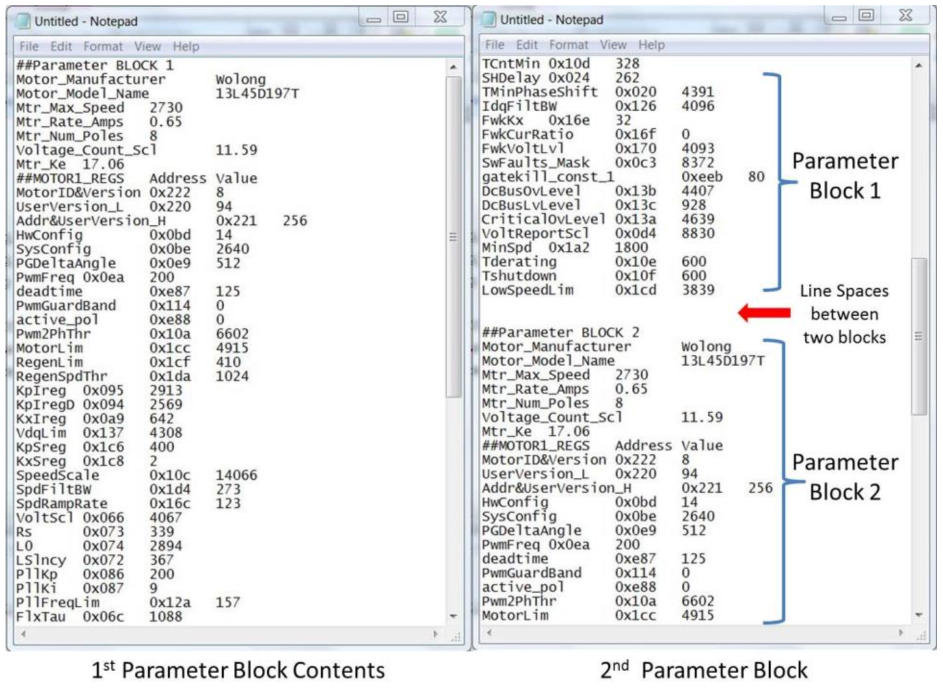
Caution must be employed while programming last parameter block in a system which requires multiple motor support. As shown in Figure 7 during OTP Parameter load, the MCE Firmware always reads HwConfig[10:9] bits of the last written parameter block only. Based on the status of these bits, the firmware identifies if multiple motor support is enabled/disabled. A system which requires multiple motor support must never have HwConfig[10:9] bits in its last written parameter block set to 00 i.e. Multiple Motor Support Disabled; or the firmware would only read the last written parameter block and not the others. For e.g. if a system requiring multiple motor support has 4 of its parameter blocks programmed, then HwConfig[10:9] of 4th parameter block must never be set to 00 or firmware will only read the 4th parameter block. Also care must be taken that MotorID is provided on the corresponding input channel as configured in HwConfig[10:9] bits of the last written parameter block.

### 3.1.2 Creating Parameter Blocks File

In order to program multiple parameter blocks into the OTP, a .txt file consisting of all the parameter blocks must be created. Following procedure must be followed to create a parameter block file:

1. Open a new 'untitled.txt' file in a text editor like 'Notepad'.
2. Configure the 1st motor and generate its parameter block .txt file using MCE Wizard. Copy the contents of the generated .txt file and paste them in the 'untitled.txt' file. This becomes the 1st parameter block.

3. Similarly, generate the parameter file for the 2nd motor and paste its contents on a new line below the 1st parameter block in 'untitled.txt'. Ensure to have more than one line spaces between the two parameter blocks as shown in Figure 8.
4. Similarly, generate parameter blocks for the other motors and continue placing them one after other with at least more than one line spaces between.
5. Ensure that maximum number of parameter blocks do not exceed more than 31 (IRMCK099 can accommodate only 31 parameter blocks)
6. Re-save the file with a suitable name and in the corresponding file directory.
7. Assign the path of this file while programming the parameter blocks using MCE Programmer.


Untitled - Notepad
File Edit Format View Help
Parameter Block 1
Line Spaces between two blocks
Parameter Block 2
1st Parameter Block Contents
2nd Parameter Block

```
##Parameter BLOCK 1
Motor_Manufacturer       Wolong
Motor_Model_Name         13L45D197T
Mtr_Max_Speed   2730
Mtr_Rate_Amps   0.65
Mtr_Num_Poles   8
Voltage_Count_Scl        11.59
Mtr_Ke  17.06
##MOTOR1_REGS   Address Value
MotorID&Version 0x222   8
UserVersion_L   0x220   94
Addr&UserVersion_H      0x221   256
HwConfig        0x0bd   14
SysConfig       0x0be   2640
PGDeltaAngle    0x0e9   512
PwmFreq 0x0ea   200
deadtime        0xe87   125
PwmGuardBand    0x114   0
active_pol      0xe88   0
Pwm2PhThr       0x10a   6602
MotorLim        0x1cc   4915
RegenLim        0x1cf   410
RegenSpdThr     0x1da   1024
KpIreg  0x095   2913
KpIregD 0x094   2569
KxIreg  0x0a9   642
VdqLim  0x137   4308
KpSreg  0x1c6   400
KxSreg  0x1c8   2
SpeedScale      0x10c   14066
SpdFiltBW       0x1d4   273
SpdRampRate     0x16c   123
VoltScl 0x066   4067
Rs      0x073   339
L0      0x074   2894
LSlncy  0x072   367
PllKp   0x086   200
PllKi   0x087   9
PllFreqLim      0x12a   157
FlxTau  0x06c   1088
```

```
TCntMin 0x10d   328
SHDelay 0x024   262
TMinPhaseShift  0x020   4391
IdqFiltBW       0x126   4096
FwkKx   0x16e   32
FwkCurRatio     0x16f   0
FwkVoltLvl      0x170   4093
SwFaults_Mask   0x0c3   8372
gatekill_const_1        0xeeb   80
DcBusOvLevel    0x13b   4407
DcBusLvLevel    0x13c   928
CriticalOvLevel 0x13a   4639
VoltReportScl   0x0d4   8830
MinSpd  0x1a2   1800
Tderating       0x10e   600
Tshutdown       0x10f   600
LowSpeedLim     0x1cd   3839


##Parameter BLOCK 2
Motor_Manufacturer       Wolong
Motor_Model_Name         13L45D197T
Mtr_Max_Speed   2730
Mtr_Rate_Amps   0.65
Mtr_Num_Poles   8
Voltage_Count_Scl        11.59
Mtr_Ke  17.06
##MOTOR1_REGS   Address Value
MotorID&Version 0x222   8
UserVersion_L   0x220   94
Addr&UserVersion_H      0x221   256
HwConfig        0x0bd   14
SysConfig       0x0be   2640
PGDeltaAngle    0x0e9   512
PwmFreq 0x0ea   200
deadtime        0xe87   125
PwmGuardBand    0x114   0
active_pol      0xe88   0
Pwm2PhThr       0x10a   6602
MotorLim        0x1cc   4915
```

Figure 8 Creating Parameter Block File

The JTAG headers on the Programming Board can interface to IRCable. An IRCable takes approximately 15 secs to program and verify 16KB of firmware into OTP.

Programming Pins

OTP programming is performed through the standard JTAG interface available on the IRMCK099 series IC. In addition to this interface, a supply voltage of 6.75V (min 6.5V and max 7V) must be supplied to the OTP memory during programming (which can be provided by the IRMCS-ISO V3.0 isolation board). This voltage is supplied to the pin VPP. When 6.75V is supplied to this pin it will act as the supply rail for the internal OTP memory.

Refer to Section 3.3.1 for the recommended procedure when programming OTP memory on a reference design board.

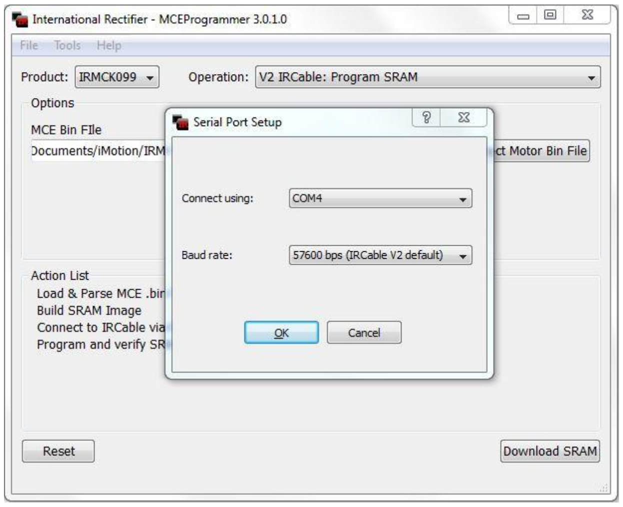
## 3.2 Using MCEProgrammer

This section gives an overview of using MCEProgrammer which is an application used to program IRMCK099.

### 3.2.1 Initial Setup for IRCable V2

IRCable V2 requires a driver for the CP2102 USB to UART bridge device used for connection to the PC. The driver is installed automatically as part of the IR development kit.

Before using IRCable V2 for the first time, the serial port must be set up in MCEProgrammer. To do this, select Serial Port Setup from the Tools menu, as shown in Figure 9. Choose the COM port that corresponds to the CP2102 USB to UART device. Make sure the default baud rate (57,600 bps) is selected. Then click OK.



Product: IRMCK099

Operation: V2 IRCable: Program SRAM

Options

MCE Bin FIle

Documents/iMotion/IRM

Action List

Load & Parse MCE .bir

Build SRAM Image

Connect to IRCable via

Program and verify SR

Serial Port Setup

Connect using: COM4

Baud rate: 57600 bps (IRCable V2 default)

**Figure 9. Serial Port Setup**

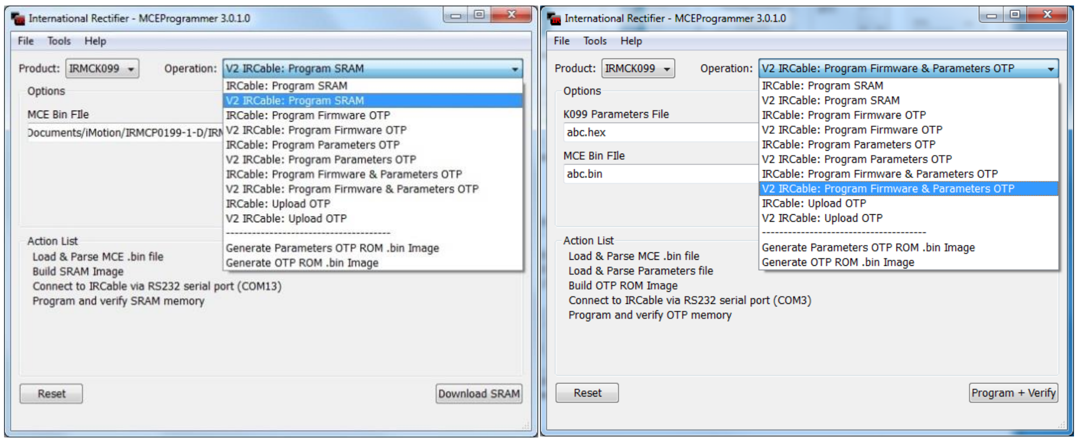
After the setup described in this section has been completed MCEProgrammer is ready for use. This step does not need to be repeated unless a change is needed.

### 3.2.2 Option Selection & Programming

All communication with the user is done through the graphical interface shown in Figure 9.

At the main window, make the following selections:

- Select the "Product" name from the pull-down list (e.g. IRMCK099).
- Select the "Operation", "V2 IRCable:Program SRAM" or "V2 IRCable: Program Firmware OTP" or "V2 IRCable: Program Firmware & Parameters OTP".
- Provide the locations of the MCE .bin or parameter.txt file you want to program.


International Rectifier - MCEProgrammer 3.0.1.0
File Tools Help
IRCable: Program SRAM
V2 IRCable: Program SRAM
IRCable: Program Firmware OTP
V2 IRCable: Program Firmware OTP
IRCable: Program Parameters OTP
V2 IRCable: Program Parameters OTP
IRCable: Program Firmware & Parameters OTP
V2 IRCable: Program Firmware & Parameters OTP
IRCable: Upload OTP
V2 IRCable: Upload OTP
Generate Parameters OTP ROM .bin Image
Generate OTP ROM .bin Image
Reset
Download SRAM
Program + Verify

Product: IRMCK099 Operation: V2 IRCable: Program SRAM

Options

MCE Bin FIle

Documents/iMotion/IRMCP0199-1-D/IRM

Action List
Load & Parse MCE .bin file
Build SRAM Image
Connect to IRCable via RS232 serial port (COM13)
Program and verify SRAM memory

Product: IRMCK099 Operation: V2 IRCable: Program Firmware & Parameters OTP

Options

K099 Parameters File

abc.hex

MCE Bin FIle

abc.bin

Action List
Load & Parse MCE .bin file
Load & Parse Parameters file
Build OTP ROM Image
Connect to IRCable via RS232 serial port (COM3)
Program and verify OTP memory

**Figure 10. MCEProgrammer Settings for IRMCK099**

After all the options are set, click “Program + Verify”.

If the “Program + Verify” option is selected, MCEProgrammer reads back the OTP content after programming completes to verify that the memory was programmed correctly. If the verification fails, an error window appears giving the address of the first byte with incorrect data.

Use the “Verify” button to read back and verify an IC that has already been programmed.

## 3.3Programming Using IRCable

The IR reference design boards provide the option of on-board programming using IRCable V2. With this option it is not necessary to use a separate board to program OTP memory. This provides a low cost “getting started kit”.

### 3.3.1 Special Procedure for OTP Programming On-Board

On the reference design board VPP must be applied to the chip in order to program OTP memory. VPP should not be applied for extended lengths of time. Therefore; the isolation box has a switch that connects VPP to the IC.

The procedure to program an IRMCK099 IC on-board is described below, which shows the IRMCS0199-1-M1 reference design board as an example. The programming procedure is the same with other reference design boards.

**Step 1. Connect the IRCableV2**

Connect the JTAG cable from IRCableV2 to the reference board. Also connect the USB cable from the IRCableV2 to the computer.

**Step 2. Turn on power**

Turn on the main power to the board. Ensure the DC-link voltage is high enough for the power supply to operate ($V_{DC}$ > Vmin). Vpp 6.75V is available through the IRCableV2. Turn on the switch while programming.

**Step 3. Open MCEProgrammer**

Open MCEProgrammer and select settings as described in section 3.2.3.

**Step 4. Apply programming voltage**

Apply programming voltage (VPP) by setting VPP switch to the ON position. The LED will light when VPP is applied. In the default setup, no external circuits will be harmed on the IC when VPP is applied.

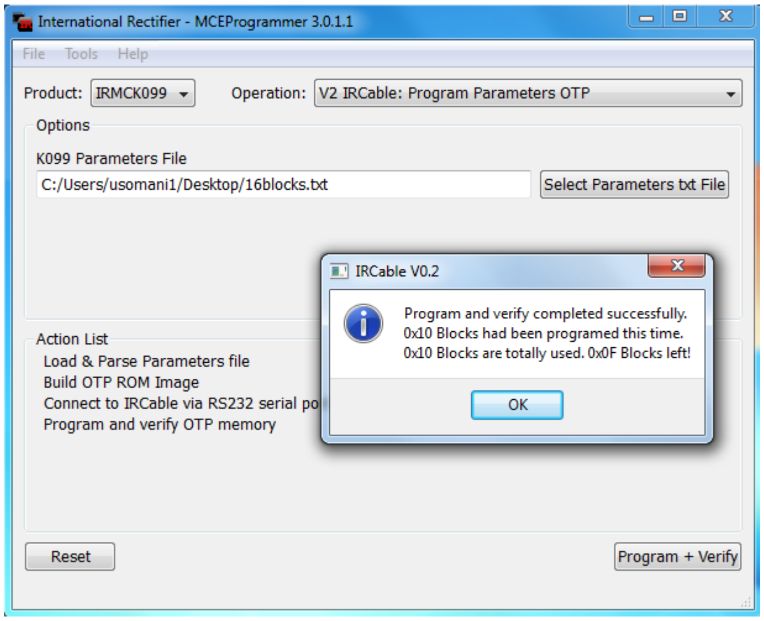
**Step 5. Programming**

Click the “Program” or “Program & Verify” button in MCEProgrammer.

The OTP of IRMCK099 is divided in 3 subsections i.e. Firmware section, Parameter Block Section and Factory Trim Section. Out of the three, factory trim section of the OTP is not accessible to application developer. MCEProgrammer offers following programming options for Firmware and Parameter Block sections of the OTP:

1. V2 IRCable: Program Firmware OTP: In this programming mode, only the firmware section of the OTP is programmed using IRCable. At the end of the write cycle, the tool performs a checksum to verify the accuracy of the programmed contents.
2. V2 IRCable: Program Parameters OTP: In this programming mode, only the Parameter Block section of the OTP is programmed using IRCable. At the end of the write cycle, the tool verifies the accuracy of the programmed contents and displays:
   a. Number of parameter blocks programmed in this write cycle.
   b. Total number of parameter blocks out of 31 already used in the OTP.
   c. Total number of parameter blocks out of 31 left available in the OTP.

   Figure 11 shows an example where parameter file with 16 (0x10) parameter blocks is programmed in OTP of an IRMCK099 in which none of the parameter blocks are programmed. Thus after the end of the write cycle, the tools displays 16(0x10) blocks have been programmed this time, 16 + 0(previously programmed) = 16 (0x10) blocks are totally used and 31-16= 15 (0x0F) blocks are left.


International Rectifier - MCEProgrammer 3.0.1.1
File
Tools
Help
Options
Select Parameters txt File
Action List
Reset
Program + Verify
IRCable V0.2
OK

Product: IRMCK099
Operation: V2 IRCable: Program Parameters OTP
K099 Parameters File
C:/Users/usomani1/Desktop/16blocks.txt

Load & Parse Parameters file
Build OTP ROM Image
Connect to IRCable via RS232 serial po
Program and verify OTP memory

Program and verify completed successfully.
0x10 Blocks had been programed this time.
0x10 Blocks are totally used. 0x0F Blocks left!

Figure 11V2 IR Cable: Program Parameters OTP

3. V2 IRCable: Program Firmare & Parameters OTP: In this programming mode, the Firmware as well as Parameter Block section of the OTP is programmed using IRCable. At the end of the write cycle, the tool verifies the accuracy of the programmed contents and displays:
   a. Number of parameter blocks programmed in this write cycle.
   b. Total number of parameter blocks out of 31 already used in the OTP.
   c. Total number of parameter blocks out of 31 left available in the OTP.

**Step 6. Turn off VPP**

When Programming/Verify completes, turn off VPP immediately by setting the switch to the off position. The VPP LED will turn off.

**Step 8. Turn off power**

Turn off the main power.

**Step 9. Remove JTAG connector**

When the DC link is discharged, remove the JTAG connector. Programming is now complete and the board can be used.

When designing the final target board (final application board without unnecessary JTAG isolation and communication related circuits), ensure that the board contains appropriate connectors to interface with the IRMCS-ISO V3 isolation board to support programming and UART/JTAG communication. The programming procedure for this case is the same as described above.

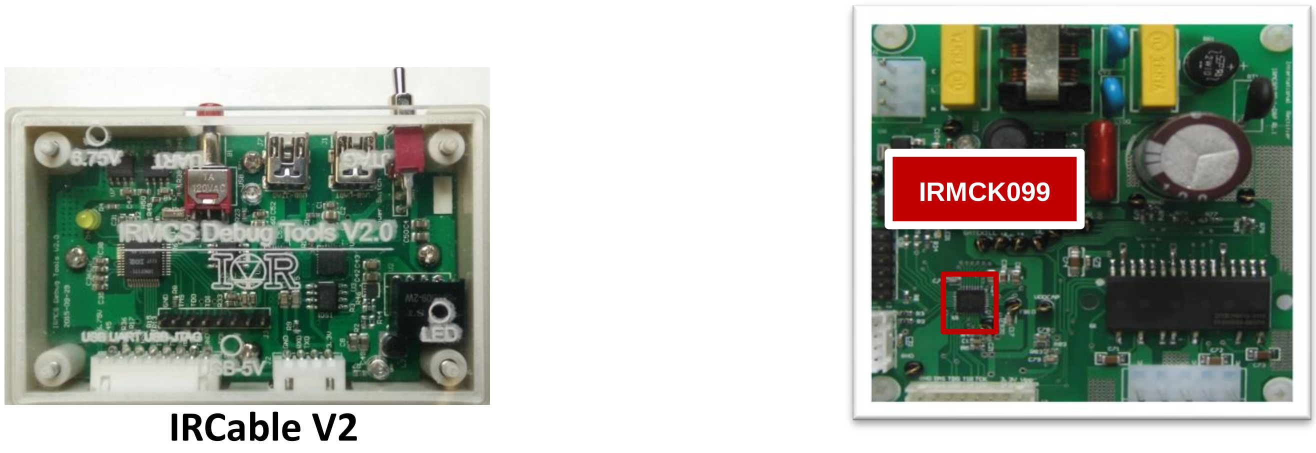
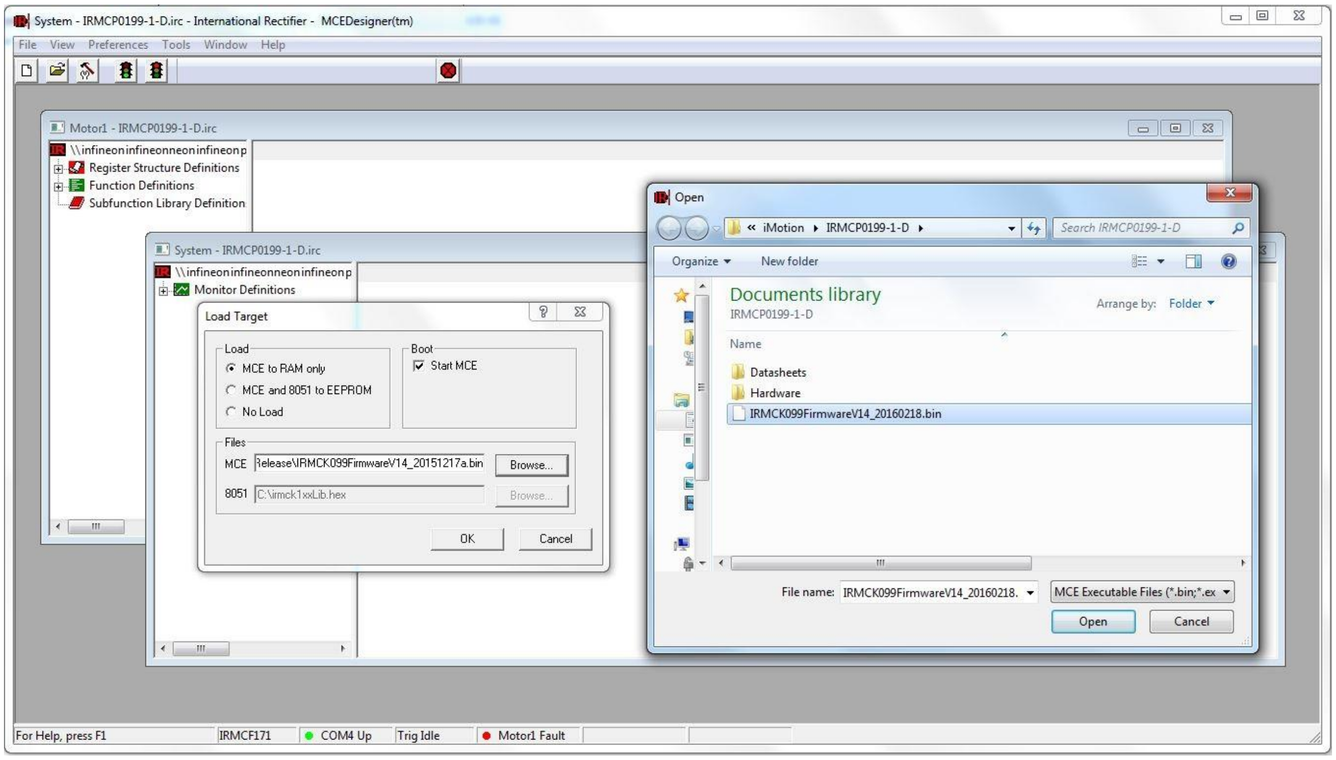


**Figure 12**. **Programming Using IRCable**

## 3.4 Download to RAM

MCEDesigner provides reload of MCE program RAM for all IRMCK099 parts. With this feature the user can download a compiled MCE design (.bin file) to the controller without having to go through the steps required in OTP programming. During development this is a fast and convenient way of programming and debugging a design. Note, however, that a program downloaded to RAM is lost on a hardware reset or when power is turned off.

Kindly refer to IRCable V2 user manual for step by step process of downloading the firmware into RAM.



System - IRMCP0199-1-D.irc - International Rectifier - MCEDesigner(tm)

Motor1 - IRMCP0199-1-D.irc
\\infineoninfineonneoninfineonp
Register Structure Definitions
Function Definitions
Subfunction Library Definition

System - IRMCP0199-1-D.irc
\\infineoninfineonneoninfineonp
Monitor Definitions

Load Target
Load: (●) MCE to RAM only ( ) MCE and 8051 to EEPROM ( ) No Load
Boot: [✓] Start MCE
Files
MCE: Release\IRMCK099FirmwareV14_20151217a.bin
8051: C:\irmck1xxLib.hex

Open: iMotion ▸ IRMCP0199-1-D
IRMCK099FirmwareV14_20160218.bin
File name: IRMCK099FirmwareV14_20160218.
MCE Executable Files (*.bin;*.ex

For Help, press F1 | IRMCF171 | COM4 Up | Trig Idle | Motor1 Fault

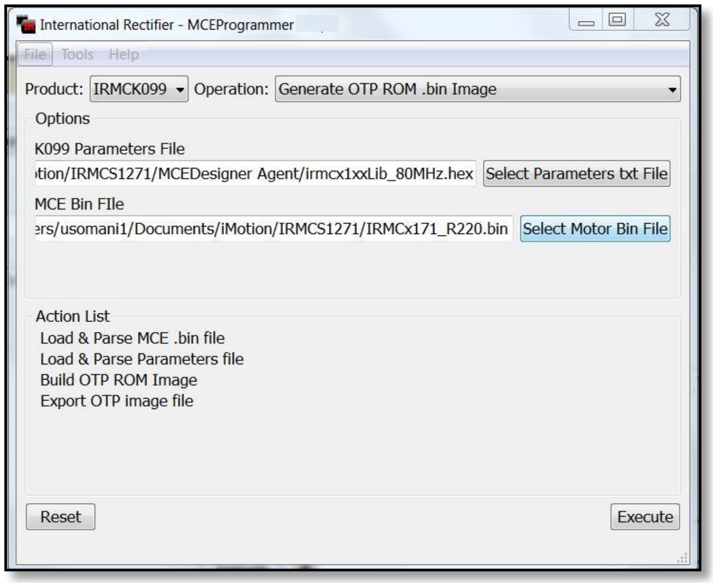
Figure 13 Download to RAM using MCEDesigner

## 3.5 Custom Programming Methods

IR provides programming tools to aid in programming the OTP. If the user chooses to design a custom programming method, detailed specifications are given in this section.

**OTP Image Generation**

The MCEProgrammer utility can be used to create IRMCK099 series OTP memory images. These binary files can then be integrated with any custom OTP programmer. The binary file created contains the address range 0x0000 to 0x2FFF for the firmware and parameter blocks from 0x3000 to 0x3F7F.The 0x3F80 to 0x3FFF address range is reserved for factory information.


International Rectifier - MCEProgrammer

Product: IRMCK099 Operation: Generate OTP ROM .bin Image

Options

K099 Parameters File
ıtion/IRMCS1271/MCEDesigner Agent/irmcx1xxLib_80MHz.hex

MCE Bin FIle
ers/usomani1/Documents/iMotion/IRMCS1271/IRMCx171_R220.bin

Action List
Load & Parse MCE .bin file
Load & Parse Parameters file
Build OTP ROM Image
Export OTP image file

**Figure 14. Generate .bin image for Firmware and parameter blocks**

## JTAG Test Mode

The IRMCK099 series has a single JTAG port that can be used to either download to SRAM or to program the OTP memory. To program the memory the controller must be put into "Test Mode". Once in this mode the OTP memory will be available over the JTAG interface.

### Programming Pins

OTP programming is performed over the standard JTAG interface available on the IRMCK099 series IC. In addition to this standard interface, a supply voltage of 6.75V must be supplied to the OTP during programming. This voltage is supplied to the pin VPP. When 6.75V is supplied to this pin it will act as the supply rail for the internal OTP memory.

During OTP read mode, the VPP pin can be at either VDD1 or VSS, or floating. When OTP is configured into program mode, the VPP pin has to be high at least 10ns before the rising edge of TCK (JTAG clock input pin). When OTP is configured back to read mode, the VPP pin has to be high at least 10ns after the rising edge of TCK, as shown in the timing diagram below. VPP high requires a typical 6.75V supply voltage, with 6.5V minimum and 7.0V maximum.

Once OTP programming is complete, the 6.75V supply voltage should be removed. Keeping VPP high for longer than about 50 seconds can damage OTP memory.

## JTAG Overview

The JTAG interface in the IRMCK099 series is the standard four-pin configuration. Data is shifted into TDI and shifted out of TDO. The state machine is controlled via the TMS line and TCK is the clock for communication. The pins are summarized in **Table 7**. Minimum recommended JTAG Clock frequency for the program and verify operation is 500 KHz.

| Pin Name | TCK | TMS | TDI | TDO |
|---|---|---|---|---|
| Function | Clock | State Machine | Serial Data In | Serial Data Out |
| Direction | Input | Input | Input | Output |

**Table 7. JTAG Pins**

### TCK Cycle

Over the course of the JTAG programming cycle, data should be loaded into TMS and TDI on the negative edge of TCK. TDO will change output states on the negative edge of TCK. Data is sampled from the TMS and TDI lines on the positive edges of TCK.

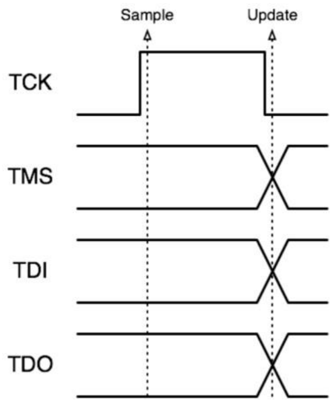


**Figure 15. Basic JTAG Cycle**

### JTAG Registers

Two registers are present in JTAG implementations, IR (Instruction Register) and DR (Data Register). The JTAG interface shifts in and out the IR and DR registers, respectively, as it performs functions. To perform a command an instruction code is loaded into IR, and then the data associated with that command is loaded into DR. The order of IR and DR loads is dependent on the type of command being performed.

There are additional registers defined for performing burn and verify operations listed in **Table 8.** These registers can be written and read with specific IR command codes. The codes used to write the registers are defined in the "IR Write Commands" section, below. Command codes used to read the registers are defined in the "IR Read Commands" section.

It should be noted that the DR register is not a physical register, but can treated as one with respect to how it behaves in the system. More information can be found in IEEE Std 1149.1.

| Register | Width | Address | Description |
|---|---|---|---|
| IR | 8 bits | - | Instruction Register |
| DR | 8,16 or 32 Bits | - | Data Register - Width depends on the instruction |
| OTP_Setup | 32 bits | 0x11 | The OTP programming configuration register |
| OTP_Wr_Timer | 32 bits | 0x12 | The OTP Timer register |
| OTP_JTAG_Address | 16 bits | 0x4eba | The current OTP address that will be accessed |
| OTP_Data | 8 bits | 0x4ebc | The data byte read from or written to OTP |
| OTP_Wait_States | 16 bits | 0x4ebe | The OTP Wait states to be written for read or write to OTP |

**Table 8. The OTP Programming Registers**

### OTP_Setup

The 8-bit OTP_Setup register contains configuration information for accessing the OTP. It is defined as shown in Table 9. The OTP_Setup register is written using IR command code 0x7a and read using command code 0x79.

| Clock Divide | | Clock Enable | PCEB | POEB | PTM | | |
|---|---|---|---|---|---|---|---|
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

| | | |
|---|---|---|
| OTP_Setup.7 – OTP_Setup.6 | ClkDiv | 01 - Sysclk/2<br>10 - Sysclk/3<br>11 - Sysclk/4 |
| OTP_Setup.5 | ClockEnable | Set this bit to1 to enable OTP clock for programming and verify |
| OTP_Setup.5 | PCEB | OTP Chip Enable bit |
| OTP_Setup.3 | POEB | OTP Output Enable Bit<br>0: Enables OTP memory read access<br>1: Enables OTP memory write access<br>This bit must be set to 1 for OTP programming |
| OTP_Setup.2 – OTP_Setup.0 | PTM | Program Test Mode<br>000 = Read OTP<br>010 = Program OTP |

**Table 9. OTP_Setup Register Definition**

### OTP_Wr_Timer

This register adjusts how long an OTP write operation will last. The write time is OTP_Wr_Timer X 64 TCK cycles. This value then is dependent on the chosen TCK rate and minimum OTP write time. The OTP_Wr_Timer register is written using IR command code 0x7a and read using command code 0x79.

### OTP_JTAG_Address

This 16-bit register is the address that the next OTP data write or read command will access. The OTP address in the register is auto-incremented after each write or read. The OTP_JTAG_Address register is written using IR command code 0x84 and read using command code 0x74.

### OTP_Data

This 8-bit register is the data that the next OTP data write command will write to the address specified in OTP_JTAG_Address. It is not necessary to access this register directly when using auto-increment operations, but it can be written using IR command code 0x84 and read using command code 0x76.

### Test_Modes

The 16-bit Test_Modes register defines what specific functions are to be performed by the test interface of the IRMCK099 series controller. Only one mode is needed for OTP programming, which is entered by writing 0x0004 into this register using IR command code 0x70. This mode sets the TCK clock input as the main system clock, which allows synchronization between the control interface and the OTP memory.

### IR Write Commands

| IR Command | Command Description |
|---|---|
| 0xF5 | Enter the ‘test mode’ for OTP memory access |
| 0xF6 | Exit the ‘test mode’ and return to standard  JTAG interface |
| 0x70 | Write contents of DR Test_Modes |
| 0x7a | Write contents of DR to OTP_Setup & OTP_Wr_Timer |
| 0x78 | Write contents of DR to OTP_JTAG_Address |
| 0x84 | Write contents of DR to Memory, 16 bit |
| 0x83 | Write contents of DR to Memory ,8bit |

**Table 10.  IR Write Command List**

In order to perform these write commands the following actions should be taken.

Step 1:

Load IR 78 - Set Access Address

Step 2:

Load DR  - Write the actual address

Step 3:

Load IR - Write or Read that address

Step 4:

Load DR data for write

Load DR 0x0 (8 bit read) , DR 0x0 0x0(16 bit read) , DR 0x0 0x0 0x0 0x0 (32 bit read)

System will auto-execute instruction with new DR

The OTP_data register can be written using IR 0x84 and the address is auto-incremented when it is being written or read. The auto-increment write mode reduces the number of commands needed during programming. When 0x84 is set in IR every following DR load will cause that data to be written to the OTP and then automatically increment the OTP_JTAG_Address register until the memory is fully programmed.

The “OTP Programming Example” section below shows an example command sequence for OTP programming.

**IR Read Commands**

| IR Command | Command Description |
|---|---|
| 0x79 | Read OTPSetup & OTP_Wr_Timer to DR |
| 0x88 | Read OTP_JTAG_Address to DR |
| 0x76 | Read 32bit Registers |

**Table 11. IR Read Command List**

When the IR register is loaded with a read command it will return the specified register value to the DR register. This value can then be returned by reading DR and shifting it out over the TDO line.

To auto-increment the OTP_JTAG_Address register for reading, a dummy byte write is required using IR 0x83 followed by DR 0x0. This improves the efficiency of memory read-back for verification after programming.

There is a latch on data read from OTP memory, so each DR read returns the data latched on the previous access. Therefore, whenever the OTP_JTAG_Address register is modified, the next OTP data read from DR is invalid and should be discarded.

The “OTP Verification Example” section below shows an example command sequence for OTP verification.

## TMS State Machine Diagram

This diagram describes the standard JTAG state machine. Through use of only the TMS signal line both IR and DR values can be loaded into the system. For more information about JTAG refer to IEEE Std 1149.1.

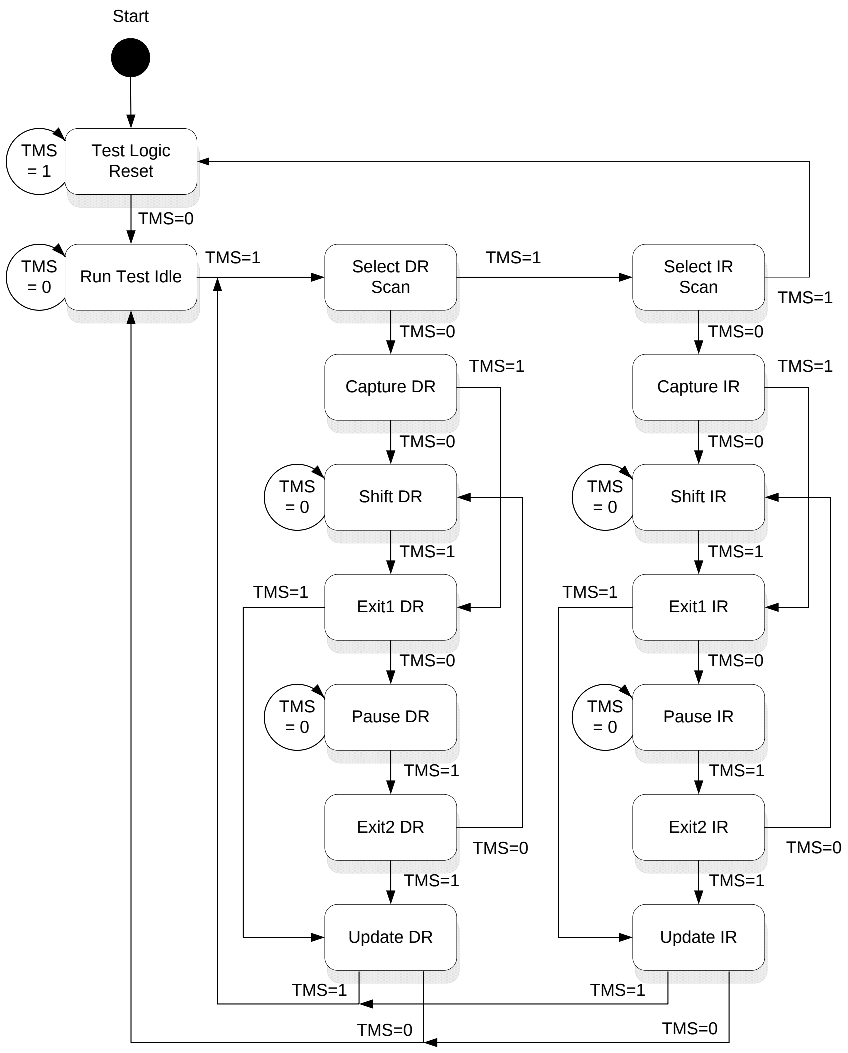


**Figure 16. JTAG TMS State Diagram**

## OTP Programming Example

```
IR 0xf5                                   // Enter JTAG Mode

IR 0x78
DR 0x18 0x30                          // Control Register
IR 0x84
DR 0x10 0x00                          // Halt MCE

IR 0x70                                   //Write or Read Test Mode Registers
DR 0x04 0x00                          //Enable  OTP Programming Mode
IR 0x78
DR 0x12 0x00                          //0x12, OTP_Wr_Timer
IR 0x7a                                   //Write 32 bit to OTP Registers
DR 0x9d 0x0 0x0 0x0                       // 0x9d

IR 0x78
DR 0x11 0x0                               //0x11, OTP_Setup
IR 0x7a
DR 0xa8 0x0 0x0 0x0                       //0xa8 to OTP_Setup
IR 0x78
DR 0x11 0x0                               //0x11, OTP_Setup
IR 0x7a
DR 0xaa 0x0 0x0 0x0                       //0xaa to OTP_Setup

IR 0x78
DR 0xba 0x4e                          //0x4eba, OTP_Addr
IR 0x84
DR 0x0 0x0 0x0 0x0                        //Starting Programming Address ,0x0000
IR 0x78
DR 0xbc 0x4e                          //0x4eba, OTP_Data
// Data write can be done in a loop and the address is automatically incremented
```

*IR 0x83* *//Write byte data*
*DR 0xData* *//Actual data, 8 bits*
*IR 0x83* *//Write byte data*
*DR 0xData* *//Actual data, 8 bits*
*IR 0x83* *//Write byte data*
*DR 0xData* *//Actual data, 8 bits*

```
....................
.....................
// Programming Complete & Restoring OTP Setup for performing verification
IR 0x78
DR 0x11 0x0                               //0x11, OTP_Setup
IR 0x7a
DR 0xba 0x0 0x0 0x0                       //0xba to OTP_Setup
IR 0x78
DR 0x11 0x0                               //0x11, OTP_Setup
IR 0x7a
DR 0xb8 0x0 0x0 0x0                       //0xb8 to OTP_Setup
IR 0x78
DR 0x11 0x0                               //0x11, OTP_Setup
IR 0x7a
DR 0x0 0x0 0x0 0x0                        //0x0 to OTP_Setup
IR 0x70
DR 0x0 0x0 0x0 0x0                        //Exit OTP Programming Mode
IR 0x78
DR 0x12 0x00                          //0x12, OTP_Wr_Timer
IR 0x7a                                   //Write 32 bit to OTP Registers
DR 0x0 0x0 0x0 0x0                        // Reset OTP_Wr_Timer
IR 0xf6                                   // Exit JTAG Mode
```

## OTP Verification Example

```
IR 0xf5                          // Enter JTAG Mode

IR 0x78
DR 0x18 0x30                 // Control Register
IR 0x84
DR 0x10 0x00                 // Halt MCE

IR 0x78
DR 0xba 0x4e                 //0x4eba, OTP_Addr
IR 0x84
DR 0x0 0x0                       //Starting Reading Address ,0x0000

IR 0x78                          // Set Access Address Register
DR 0xbe 0x4e                 // 0x4ebe, OTP_WAIT_STATES Register
IR 0x84                          // Write 16 bit data to 0x4ebe
DR 0x01 0x00                 // 0x01 Wait_States
IR 0x78
DR 0xbc 0x4e                 //0x4eba, OTP_Data
// Data read can be done in a loop and the address is automatically incremented
.....................
.....................
.....................
IR 0x83                          //Write byte data
DR 0x0                           //Dummy Write to do the auto increment the address
IR 0x76
DR 0x0 0x0 0x0 0x0
IR 0x83                          //Write byte data
DR 0x0                           //Dummy Write to do the auto increment the address
IR 0x76
DR 0x0 0x0 0x0 0x0

......................
......................
......................
......................

IR 0xf6                          //Exit JTAG Mode
```

## OTP IR Load Example

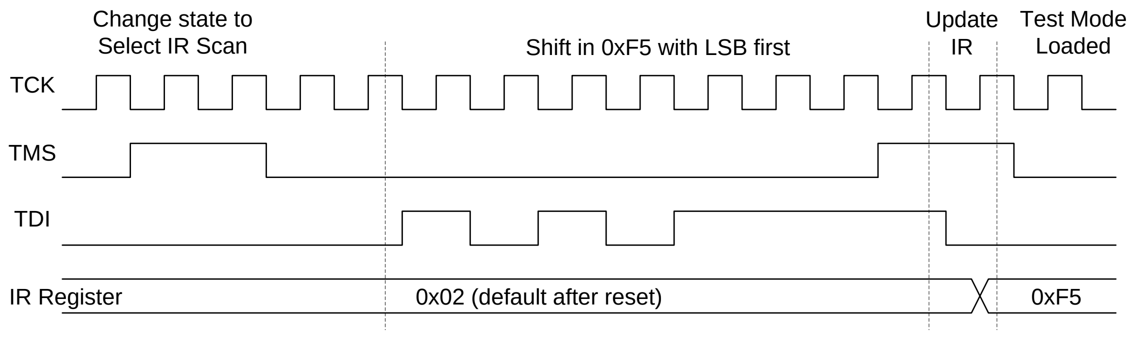

Change state to Select IR Scan
Shift in 0xF5 with LSB first
Update IR
Test Mode Loaded
TCK
TMS
TDI
IR Register
0x02 (default after reset)
0xF5

**Figure 17. JTAG Load 0xF5 to IR (Enter Test Mode)**

**OTP Timing Information**

There are three important parameters required when programming the OTP through the JTAG interface. This information is listed below.

| Parameter | Symbol | Min | Max | Unit |
|---|---|---|---|---|
| Clock Cycle Time | $T_{cyc}$ | 25 | - | ns |
| Program Pulse Width | $T_{pw}$ | 100 | Note 1 | µs |
| Program Pulse Interval | $T_{pwi}$ | 5 | Note 1 | µs |

Note 1. There is no specific maximum pulse width or pulse interval, but the total programming time should be kept under 50 seconds since operation with VPP high for longer periods can damage OTP memory. VPP is not required for reading the OTP.

**Table 12. Critical OTP Programming Timing Information**

**Edition 2016-03-16**

**Published by**

**Infineon Technologies AG**

**81726 München, Germany**

**Do you have a question about this document?**

**Email: erratum@infineon.com**

**Document reference**

**ifx1**