Software Developer's Guide

iMOTION™ motor control IC with additional MCU

About this document

Scope and purpose

The IRMCx100 series motor control ICs are mixed signal devices optimized for permanent magnet motor control. They combines the iMOTION™ motion control engine (MCE) with an additional 8 Bit microcontroller (MCU) to improve application flexibility.

The MCE can perform sensor less FOC over the full speed range of the motor, including providing torque at zero speed and stable control at deep field weakening speeds. The IRMCx100 series motor control ICs can be combined with a µIPM™ or a discrete power stage to implement a complete inverter for the control of a PMSM. Infineon is offering its ready-to-use and field proven FOC control algorithm along with the IRMCx100 products making it extremely easy for customers to get started.

With the IRMCx143 there is also a dedicated device available for servo drive applications.

The integrated standard 8051 MCU can be programmed by the customer to interact with the MCE and to implement additional functionality running almost independently from the motion control algorithm on the MCE. A number of peripheral devices and special functions have been added to customize the operation for motor control applications.

The 8051 and MCE interact through a shared RAM, accessible by both processors.

The purpose of this guide is to describe the implementation of 8051 microprocessor control for use in the IRMCx100 series of motion control ICs. This document covers required initializations, settings and functions for 8051 control of the IC, as well as programming of the OTP/Flash memory using Infineon's demo boards or by in-circuit programming. Some examples are presented. This application note assumes that the user has experience with embedded software programming and is also familiar with the application developer's guide and one of Infineon's reference design kits.

An additional document, the Reference Manual, also referred to frequently in this document, has detailed information on many topics covered here, as well as full descriptions of the MCE & MCU hardware registers and programming methods.

Intended audience

This software developer's guide is intended for customers implementing an inverterized drive.

User Guide #UG-10002 V2.2

IRMCx100 Software Developer's Guide

By International Rectifier's iMotion Team

Table of Contents

Table of Figures

Table of Tables

1 Introduction

The 8051 microprocessor, included in the IRMCx100 series of motion control ICs (IRMCF171, IRMCK171, IRMCF143, IRMCK143), can be used to implement a large variety of control and protection functions for motor control applications. The instruction set and basic operation of the IRMCx100 Series 8051 microprocessor is consistent with the standard Intel 8051 processor. A number of peripheral devices and special functions have been added to customize the operation for motor control applications.

The IRMCx100 series ICs contain two processors: an 8051 processor and the Motion Control Engine (MCE). The 8051 and MCE interact through a shared RAM, accessible by both processors. The MCE is designed specifically to implement motor control loops, process feedback signals, and calculate PWM switching signals. The 8051 mediates between external control signals (such as the front panel of a washing machine) and the MCE, which ultimately produces the signals that operate the motor.

Corresponding to the two processors in the IC, there are two main parts of the code required for proper operation of the IRMCx100 Series IC: the 8051 program and the MCE program. The MCE program provides the PWM synchronous operations for calculation of the proper inverter gating. A standard MCE program is supplied by IR in the form of "IRMCx1*xx*_R*nn*.bin" (where *xx* identifies the IC and *nn* identifies the version number). This program may be modified by the user, but it will have the same file extension, ".bin". The .bin file is the primary output of MCE Compiler.

The 8051 provides application level control and asynchronous interrupts to modify settings in the MCE program. This application is generally developed by the user. The 8051 program, in the format of a .hex file, is an output of the Keil uVision compiler. In this document, the MCE and 8051 code images may be referred to as the .bin file and .hex file, respectively. The .hex file is in standard Intel hex format. Information on the Intel hex record format can be found here:
http://www.keil.com/support/man/docs/oh166/oh166_ih_record.htm

The 8051 software controls and monitors the operation of the MCE through the read/write register interface of the shared RAM. The 8051 code (MCEDesigner Agent) used with the MCEDesigner tool operates in a simple lock-step manner: MCEDesigner specifies each register to be read or written individually and the 8051 software performs only those operations as they are requested. An 8051 user application, on the other hand, would typically perform entire sequences of operations automatically or in response to simple input commands such as "start" and "stop."

1.1 Purpose

The purpose of this guide is to describe the implementation of 8051 microprocessor control for use in the IRMCx100 series of motion control ICs. This document covers required initializations, settings and functions for 8051 control of the IC, as well as programming of the OTP/Flash memory using IR's DEMO boards or by in-circuit programming. Some examples are presented. This application note assumes that the user has experience with embedded software programming and is also familiar with the Application Developer's Guide and one of IR's Reference Design Kits.

The examples given here are intended to allow the designer to create a control interface to replace MCEDesigner once the application development has been completed. One of the main tasks is to recreate MCEDesigner functions in the 8051 code. The 8051 code can be written and debugged on the Flash memory version (IRMCF100) and then migrated to the IRMCK100 (OTP memory version) for production, if desired.

1.2 Requirements

The following software and hardware is required for 8051 application code development:

1. MIPS SNAV-M8051EW Debugger with Keil uVision driver (referred to as the "FS2 debug pod" in this document), available at: http://www.mips.com/products/software-tools/legacy/8051/
2. Keil PK51 Professional Developers Kit (Keil uVision2 or uVision3)
3. An IR DEMO board or IRMCS-ISO V3.0 (or higher version) isolation board and the customer designed application board.
4. 100SeriesKitMCEInstallerV2.1.exe or later version installer.

Using Keil uVision, the developer can write the 8051 control program in the C programming language and then compile it into machine code and program it to the IRMCx100 memory for testing. Keil's uVision development environment provides a simulation mode so that portions of the program which are not hardware-dependent can be tested without programming the actual IC. The sample source code, IRSamples, provided with IR's development tools, was developed with Keil uVision.

In addition to the requirements above, the circuit board containing the IRMCx100 IC should contain the appropriate connectors, drivers, and isolators to interface with the FS2 hardware. If IR's Reference Design Boards are used, the appropriate circuits are built in, with proper isolation for the FS2 hardware connection. If any other hardware is used, follow the instructions in the warning below.

Warning!
When connecting the FS2 debug pod to the circuit board, the FS2 hardware can be damaged by the high voltage on the board if appropriate isolation is not used. The problem arises because the DC bus minus (GND) is not at the same potential as Earth (or wall) ground. For this reason, if proper isolation is not used, it is recommended that the board be powered by a DC power supply with isolated ground when using the FS2 hardware.

The IRMCS-ISO V3.0 (or higher version) isolation board is needed if developing 8051 code on a non-isolated IRMCx100 user target board (final application board), to ensure that the PC is fully isolated from the AC Grid.

The isolation circuits are integrated directly on the IRMCS1043 reference design, or provided on IRMCS-ISO V3 for IRMCS1271, including all of the circuits for JTAG and UART isolation, USB to RS232 conversion and 6.75V programming voltage.

When designing an application board, ensure that it contains the right connectors for the IRMCS-ISO-V3. The isolation board and connection interface are shown Figure 1 . Please refer to the IRMCS-ISO-V3 manual for more information.

For safety from electrical shock, a 1:1 AC power isolation transformer with proper power rating is highly recommended to isolate the motor control bench.

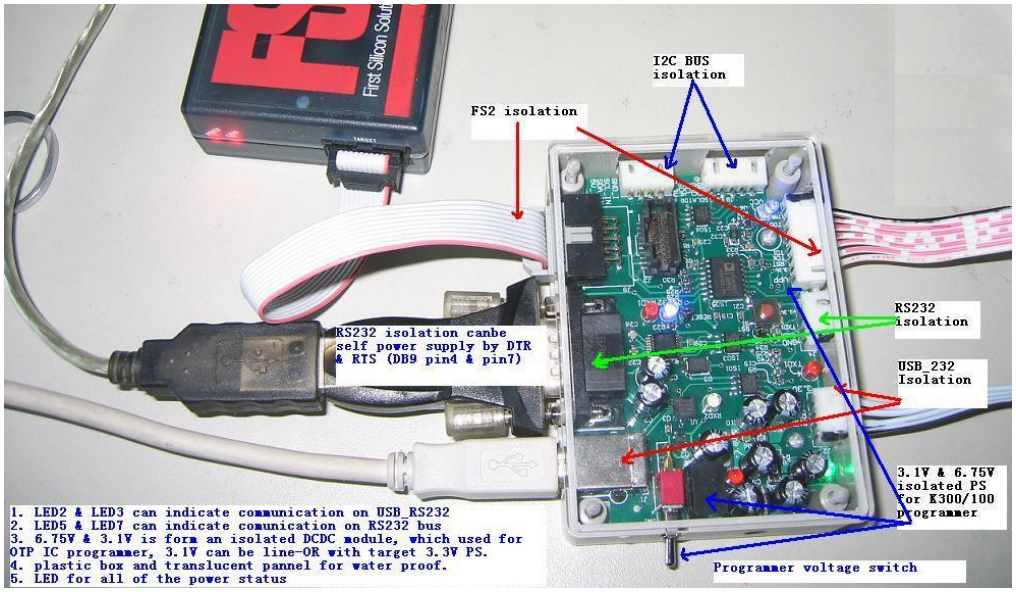


Figure 1. IRMCS-ISO V3 Isolation board and connection

1.3 Boot Process

The main task of the boot process is to copy the MCE program stored in non-volatile memory to MCE program RAM, initialize the 8051 program counter to the start of OTP or flash memory and transfer control to the 8051 CPU. For security, the MCE program is encrypted by the MCE Compiler and is stored in encrypted form in flash or OTP memory. The MCE program is decrypted during the boot process as it is copied into MCE program RAM.

The reset and boot processes are closely tied together. The boot process is automatically accomplished following a proper reset sequence. Reset of the IC is triggered by any of the following events:
1) Power up
2) Under voltage lockout (UVCC) detects low voltage on 3.3V.
3) The watchdog timer times out.
4) External reset, achieved by holding RESET pin low for a minimum of 10usec.

Once the reset is recognized in the system, the reset module counts up to 2048 clocks at the crystal frequency (i.e. 4 MHz, 512 µsec) to insure that the internal PLL becomes stable for generation of the internal system clock. When this waiting period is complete, the boot module begins copying the MCE program from OTP memory to MCE program RAM. The full 12Kb MCE program area is always copied, regardless of the actual MCE program size. The time to complete the copy process depends on the boot clock frequency, as shown below.

- Total MCE program to be copied: 12 * 1024 bytes
- Number of clock periods to transfer one byte (including synchronizers and decryption logic): 20
- Boot clock freq: 1MHz for internal watchdog clock or 4MHz as an example of an external crystal oscillator.

Total transfer time (1MHZ) = 12 * 1024 * 20 * 1 µsec = 244 msec
Total transfer time (4MHZ) = 12 * 1024 * 20 * 0.25 µsec = 61 msec

A user application cannot intervene during the reset and boot process. Immediately after the copy process completes, the 8051 application program begins execution.
For security, the MCE program is automatically encrypted by the MCE compiler and stored in OTP memory in encrypted form. The program is decrypted automatically as it is copied to MCE program RAM during the boot process. The 8051 does not have read access to MCE program RAM, so the MCE program cannot be read once it has been decrypted.

For more detail on the boot process and selection of boot clock source, please refer to the IRMCx100 Reference Manual.

1.4 8051 and MCE Internal Clocks

Because the 8051 executes its program directly from internal memory its clock rate must be constrained to the maximum operating frequency of the flash or OTP memories:

- Maximum OTP frequency for IRMCK100: 32MHz
- Maximum Flash frequency for IRMCF100: 20MHz

Because the MCE program executes from RAM it is possible to run the MCE significantly faster as long as the 8051's clock is divided. The IRMCx100 series IC supports running the MCE clock at an integer multiple the 8051 clock. For example, for IRMCK100, this means the maximum frequency of 128MHz could be maintained for the MCE, while the 8051 could run at 32MHz. This clock division is accomplished through the register PLLF2. The configuration of the clock divider is different in IRMCF100 and IRMCK100 as shown in Table 1. It is important to note that some IRMCx100 series peripheral devices and clocked components operate on the slower 8051 clock and others use the faster MCE clock. Please refer to the description of each component in the IRMCx100 Reference Manual for specific clocking information.

Component	PLLF2 Bit Field	Value	8051 Clock Frequency
IRMCK100	PLLF2.6 PLLF2.5	00	MCE Clock/1
		01	MCE Clock/2
		10	MCE Clock/3
		11	MCE Clock/4
IRMCF100	PLLF2.6 PLLF2.5 PLLF2.4	000	MCE Clock/1
		001	MCE Clock/2
		010	MCE Clock/3
		011	MCE Clock/4
		100	MCE Clock/5
		101	MCE Clock/6
		110	MCE Clock/7
		111	*Reserved*

Table 1. PPLF2 Register Options for Clock Division

For smooth transition from IRMCF1xx to IRMCK1xx, the recommended rates are:
- MCE clock frequency of 100MHz with an 8051 clock frequency of MCE clock / 5; or
- MCE clock frequency of 80MHz with an 8051 clock frequency of MCE clock / 4.

1.5 Memory Resources

The IRMCx100 series provides four memory-mapped resources:

- Non-volatile memory for program storage
- RAM for MCE program execution
- RAM for 8051 and MCE data storage
- Memory-mapped motion hardware registers (MHRs)

Access to some of these resources by the MCE and 8051 is limited, as described in Table 2, below.

Memory Resource	Size (bytes)	Access by 8051	Access by MCE
OTP Memory (IRMCK1xx)	32K	Read only	None
Flash Memory (IRMCF1xx)	64K	Read only	None
MCE Program RAM	12K	Write only	Read only
Data RAM	4K	Read and write	Read and write
Motion Hardware Registers	1K	Read and write	Read and write

Table 2. Access to Memory Resources

The 8051 is an 8-bit processor and memory is addressed in 8-bit bytes, whereas the MCE is a 16-bit processor and memory is addressed in 16-bit words. Both processors have separate program and data address spaces; it is allowable for address ranges in program space to overlap ranges in data space. In 8051 address space, the non-volatile memory (OTP or flash) is mapped to program space, while MCE program RAM, data RAM and the motion hardware registers (MHRs) are mapped to data space. In MCE address space, both program and data RAM are based at address 0x000. MCE program RAM is mapped to program space, with data RAM and the motion hardware registers (MHRs) mapped into data space.

1.5.1 OTP Memory (IRMCK1xx)

Figure 2 illustrates the 8051 and MCE address maps for the IRMCK1xx. OTP memory is shown in yellow. The corresponding figure for the IRMCF1xx is shown in the next section. Note that the 8051 program and data address spaces have overlapping address ranges.

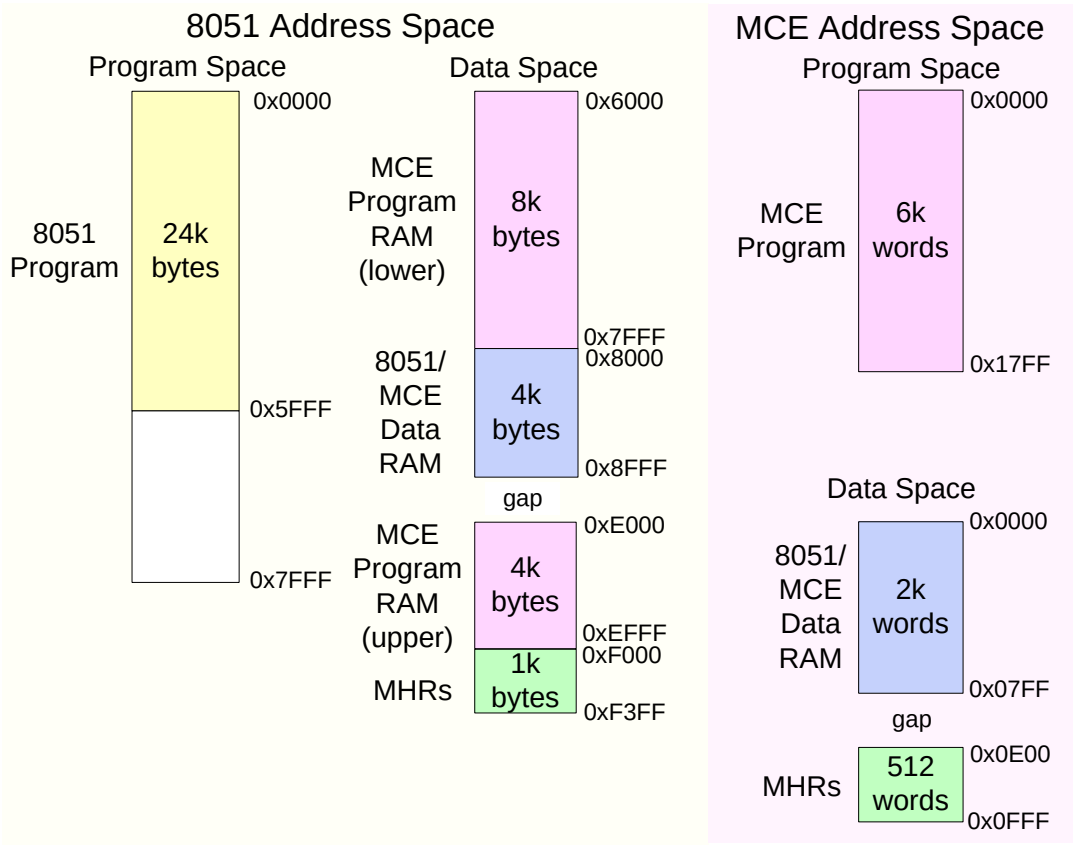


Figure 2. Memory Map of IRMCK100 Series

The MCE program is shown in pink. This 12K-byte area of memory is mapped contiguously in MCE program address space, but in the 8051 data address space, it's divided into an 8K-byte segment and a 4K-byte segment. Shared data RAM is shown in blue and the MHR area is shown in green. These areas are mapped to both 8051 and MCE data address spaces.

For the IRMCK1xx, the 32K-byte OTP memory holds both 8051 and MCE program. The 8051 program executes directly from this area of OTP memory. The MCE cannot address OTP memory. The IRMCK100-series hardware copies MCE program from OTP to MCE program RAM during the boot process and MCE program executes from RAM.

OTP memory is divided into two areas with the 8051 program contained in the first section and the MCE program contained in the second. The boundary between the two areas is dynamic and can vary from 20K bytes to 24K bytes. This allows a maximum size of 24K bytes for the 8051 program (limiting MCE program size to 8K bytes) or a maximum size of 12K bytes for the MCE program (limiting the 8051 program to 20K bytes). The total program size (8051 + MCE) cannot exceed 32K bytes. The last 8K bytes of OTP are reserved for the MCE application program and this content is not normally accessed by the 8051 CPU. (This boundary is enforced by the iMotion software tools but is not a hardware requirement. It is possible for the 8051 application program to exceed 24K bytes in size if the MCE program is smaller than 8K bytes.)

Figure 3 illustrates how OTP memory is used. Diagram A shows that the 8051 program begins at the base of OTP memory and extends toward higher offsets. MCE program begins at the top of OTP memory and extends toward lower offsets. The last 12K bytes of OTP are copied to MCE program RAM at boot time, limiting the MCE program size to a maximum of 12K bytes. The diagram shows the potential overlap of 8051 program and MCE program at OTP offsets 0x5000 – 0x5FFF. The user must ensure that such an overlap does not occur. Diagram B shows an 8051

program using the maximum 24K-byte portion of OTP memory. This limits the MCE program to the remaining 8K bytes. Diagram C shows an MCE program using the entire 12K byte area that is copied to MCE program RAM at boot time. This limits the 8051 program to the first 20K bytes. Diagrams B and C both show valid allocations of OTP memory.

The size of the MCE reference design for the IRMCx171 is just under 8K bytes and the IRMCx143 reference design is about 10K bytes. This includes the MCE firmware as well as the compiler-generated code that implements the user application. The size of the MCE program code does not include the header at the beginning of the .bin file, so the program size is slightly smaller than the total size of the .bin file.

Note that the MCE compiler displays the total MCE program size in words (16 bit units). Therefore, the displayed size must be multiplied by two to determine the program size in bytes.

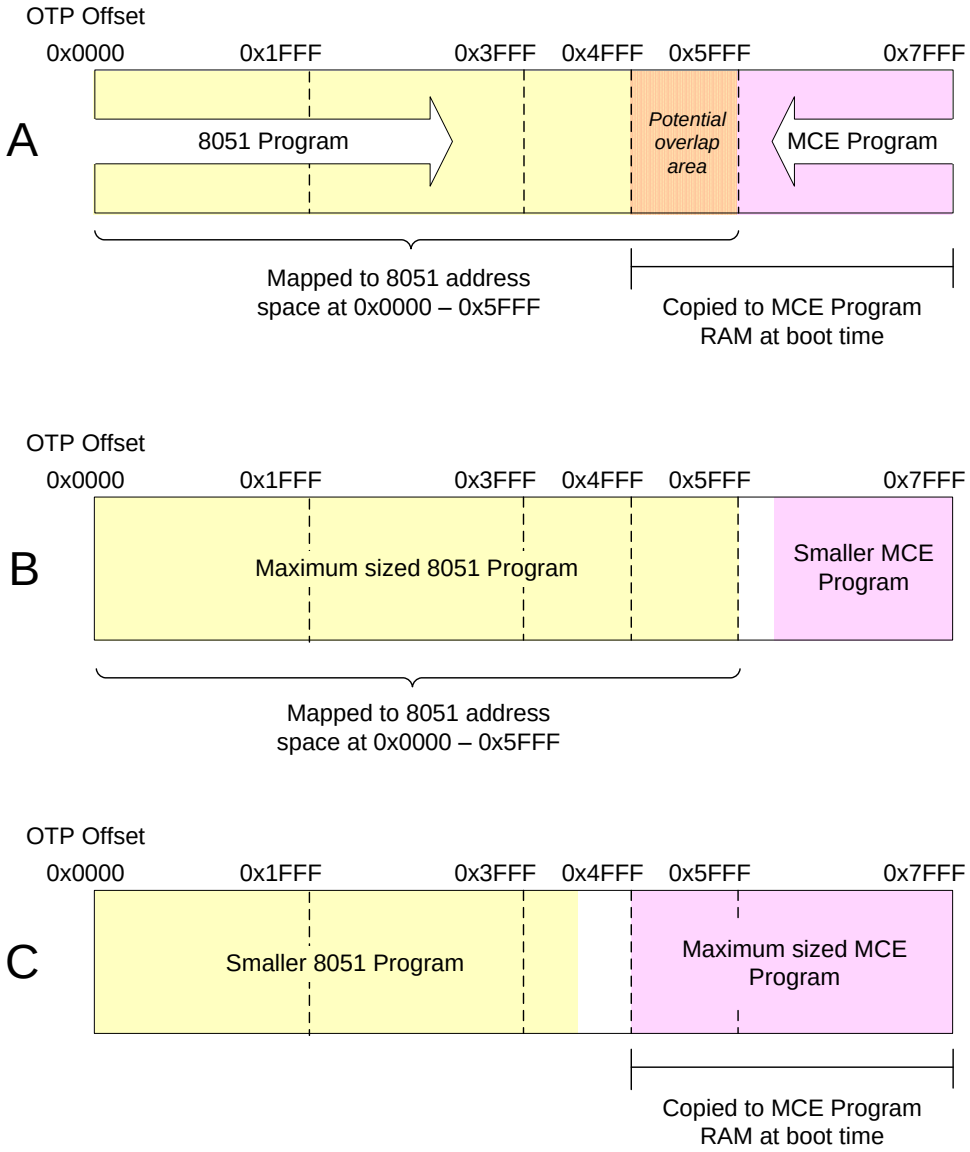


Figure 3. OTP Memory Usage for IRMCK100 Series

1.5.2 Flash Memory (IRMCF1xx)

For the IRMCF1xx, the 64K-byte flash memory holds both 8051 and MCE program. The 8051 program executes directly from this area of flash memory. The MCE cannot address flash memory. The IRMCF100-series hardware copies MCE program from flash to MCE program RAM during the boot process and MCE program executes from RAM.

The MCE program is shown in pink. This 12K-byte area of memory is mapped contiguously in MCE program address space, but in the 8051 data address space, it's divided into an 8K-byte segment and a 4K-byte segment. Shared data RAM is shown in blue and the MHR area is shown in green. These areas are mapped to both 8051 and MCE data address spaces.

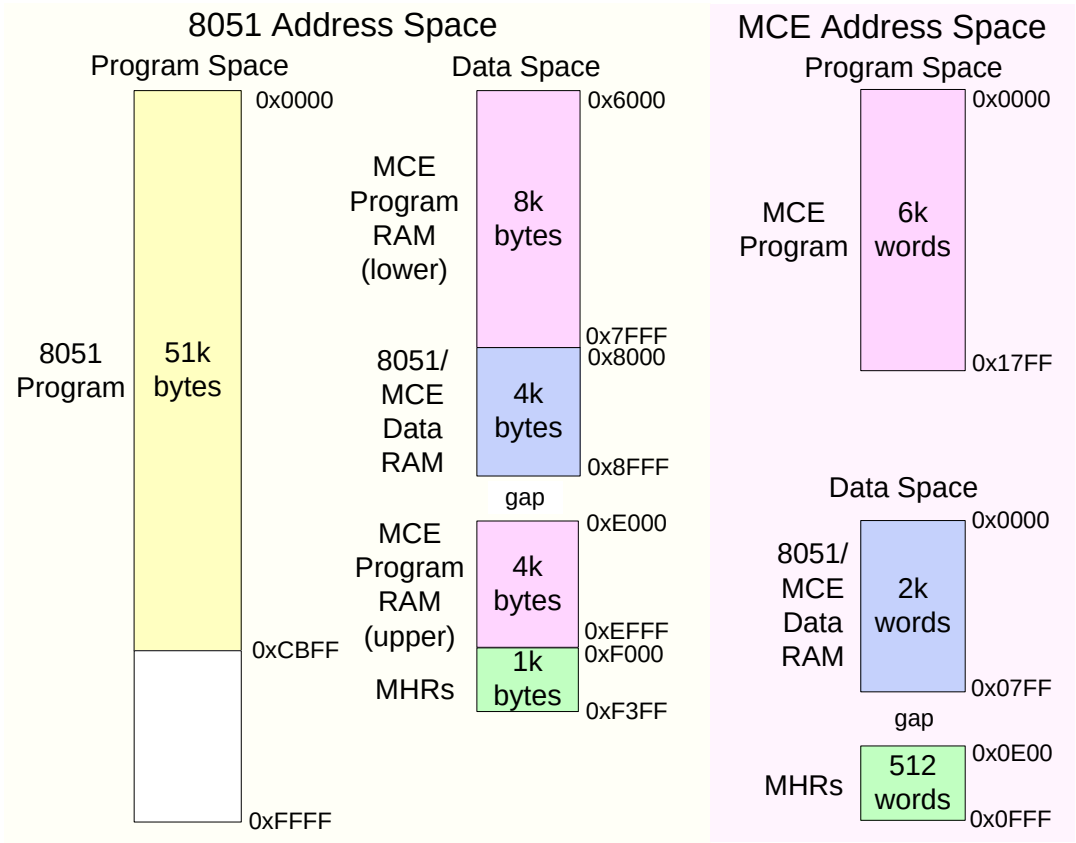


Figure 4. Memory Map of IRMCF100 Series

The 64K-byte flash memory contains both 8051 and MCE programs, and is divided into two areas with the 8051 program contained in the first section and the MCE program contained in the second. The boundary between the two areas is fixed at offset 0xCBFF and the upper 1K bytes (beginning at offset 0xFC00) are reserved. This allows a maximum size of 51K bytes for the 8051 program and a maximum size of 12K bytes for the MCE program. (This boundary is enforced by the iMotion software tools but is not a hardware requirement. It is possible for the 8051 application program to exceed 51K bytes in size if the MCE program is smaller than 12K bytes.)

Figure 5 illustrates how flash memory is used. The 8051 program begins at the base of flash memory and extends toward higher offsets. MCE program begins near the top of flash memory and extends toward lower offsets. The upper 1K bytes of flash memory are reserved. The entire flash memory is mapped to 8051 program address space. The 12K bytes of MCE program are

copied from flash to MCE program RAM at boot time. 8051 and MCE program ranges do not overlap in flash as they do in OTP memory.

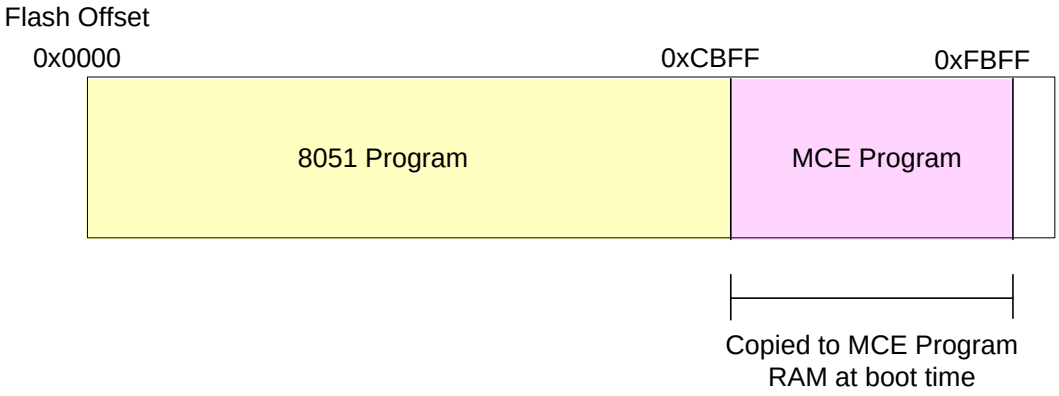


Figure 5. Flash Memory Usage for IRMCF100 Series

1.5.3 The Shared RAM

There are several methods by which the 8051 processor and MCE can communicate through share resources. These include:

- Shared RAM
- MCE processor registers
- MCE motion peripheral configuration register interface
- Interrupts from the MCE to the 8051

See the previous sections for information on how the memory resources described here are mapped to 8051 and MCE memory.

Shared data RAM for the 8051 and MCE has a total size of 4K bytes. This area is used for MCE private data storage, 8051 private data storage and shared data, such as the motion firmware registers (MFRs).

A typical allocation of shared data RAM is shown in Figure 6. The MCE compiler determines MCE firmware RAM usage dynamically. MCE firmware RAM usage can vary depending on the firmware version, so the division between MCE firmware and MCE application RAM is not fixed.

Typically about 1280 bytes are allocated for MCE firmware data at 8051 addresses 0x8000 – 0x84FF. the MCE Compiler allocates additional data for the MCE user application in the range 0x8500 – 0x85FF. These two areas of RAM are used for MCE private storage and for information passed between the MCE and the 8051. Both the 8051 and the MCE access these areas of RAM for reading and writing. The remaining area of shared RAM at addresses 0x8600 – 0x8FFF is reserved for 8051 local data storage. The MCE does not access this area.

Remember that the 8051 data range must be defined in the Keil compiler options, as noted in Section 2.1.1, below.

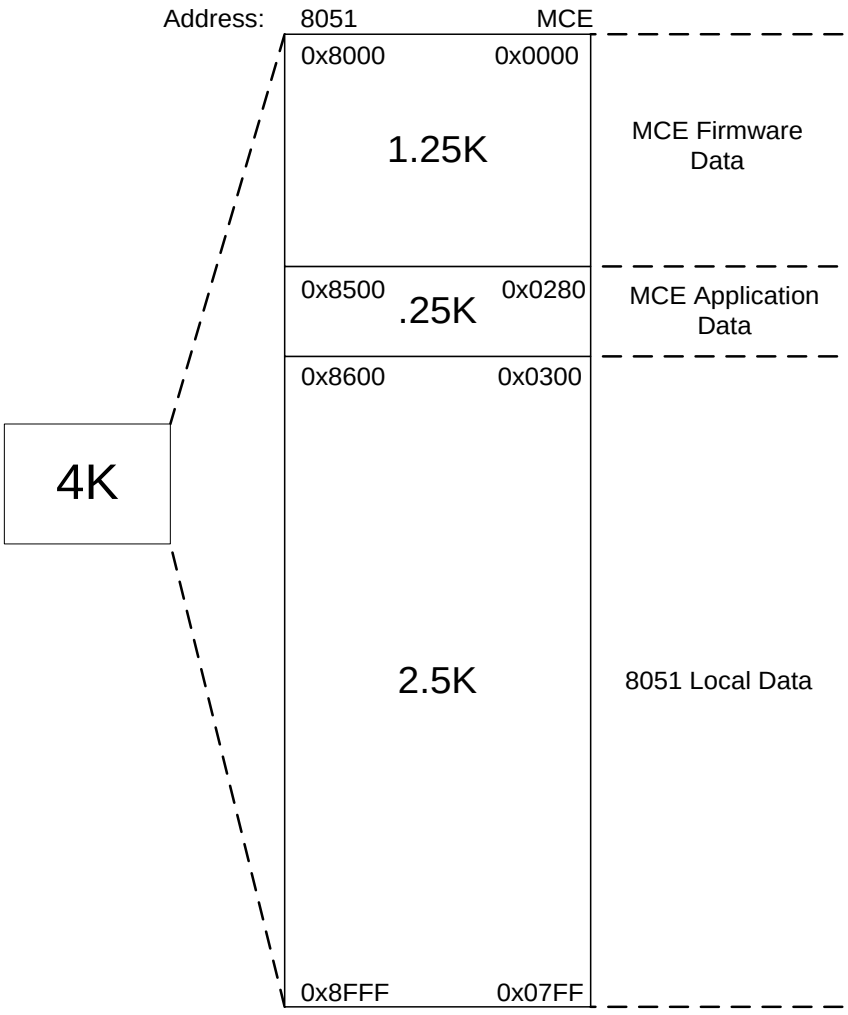


Figure 6. Typical Shared RAM Allocation for IRMCx100 Series ICs

1.5.4 8051 vs. MCE Addressing

As shown in Figure 2, the MCE has separate data and program address spaces, both of which are based at address 0x0000. MCE data address 0x0000 corresponds to the base of the shared RAM at address 0x8000 on the 8051 memory bus. MCE program address 0x0000 corresponds to the base of MCE program memory at address 0x6000 on the 8051 memory bus. In the 8051 address space, MCE program memory is divided into two segments, but in the MCE program address space, the program RAM is contiguous.

The 8051 addresses memory 8 bits (1 byte) at a time, while the MCE addresses memory 16 bits at a time, requiring half the memory addresses to access the same data space. This is reflected in the 8051 and MCE addresses shown in the memory maps above.

1.5.5 Byte Ordering

The Keil compiler used for 8051 software development generates code that uses big endian byte ordering to store 16-bit and 32-bit values in memory. The MCE is a 16-bit processor and uses little endian byte ordering for data storage. *Functions to correctly read and write the shared RAM are included in the sample code, IRSamples. These functions correctly swap bytes when*

necessary, and lock out bus accesses to prevent data being read by one processor while it is still being written by the other.

Byte ordering refers to the convention used to store 16-bit and 32-bit values in memory using a processor, such as the 8051, that has a native addressing mode of 8 bits. The two standard byte ordering conventions are “big endian” or “Motorola” byte ordering and “little endian” or “Intel” byte ordering.

In big endian byte ordering, the “big end” of a value is stored first. That is, the high order byte is stored at the lowest memory address and the low-order byte is stored at the highest memory address. In little endian byte ordering the “little end” is stored first, with the low-order byte at the lowest memory address.

For example, suppose the 16-bit value 0x2345 is to be stored in memory at address 0x1000. Using big endian byte ordering, 0x23 is stored at address 0x1000 and 0x45 is stored at address 0x1001. Using little endian byte ordering, 0x45 is stored at address 0x1000 and 0x23 is stored at address 0x1001. Table 3 below shows how the value 0x456789AB would be stored at address 0x1000 using each of the byte ordering conventions.

Address	Big Endian	Little Endian
0x1000	0x45	0xAB
0x1001	0x67	0x89
0x1002	0x89	0x67
0x1003	0xAB	0x45

Table 3. Big and little endian byte conventions

The Keil compiler used for 8051 software development generates code that uses big endian byte ordering to store 16-bit and 32-bit values in memory.

The MCE is a 16-bit processor and uses little endian byte ordering for data storage. The smallest unit of data storage on the MCE processor is 16 bits (it cannot access a single byte in memory). The shared RAM used to exchange information between the 8051 and MCE processors is 8-bit addressable to the 8051, but 16-bit addressable to the MCE.

The MCE expects all data shared between the 8051 and MCE processors to be in little endian byte ordering. This means that the 8051 must swap bytes before writing to shared RAM and swap bytes after reading from shared RAM. Table 4 below shows how the value 0x456789AB would be correctly stored by the 8051 for sharing with the MCE. Note that the 8051 reads and writes a byte at a time, but the MCE always accesses the memory a word (16 bits) at a time.

8051 Address	8051 Bytes	MCE Address	MCE Words
0x8200	0xAB	0x0100	0x89AB
0x8201	0x89		
0x8202	0x67	0x0101	0x4567
0x8203	0x45		

Table 4. Byte Ordering for Sharing of Data between 8051 and MCE

1.5.6 Synchronizing 8051 Register Access with the MCE

In user applications, the 8051 may be required to monitor or modify MCE registers during motor operation. There are some risks of errors when reading or writing to the dual-port RAM. To guard against problems, the designer can use the SYNC interrupt to synchronize 8051 access with the PWM cycles which dictate the updating of the inverter gating signal duty cycles.

The SYNC interrupt is generated from the MCE to signal the 8051 that a SYNC pulse has occurred. The SYNC interrupt is a periodic event signal generated by the MCE. Its timing is illustrated in Figure 7. This is the most important signal used for synchronization between the 8051 (CPU side) and the MCE (motion control side). An 8051 application software task that needs to pass commands to the MCE and/or receive updated data from the MCE may require specific synchronization with the MCE. This is due to the fact that MCE computation is initiated and triggered by the SYNC pulse at every PWM carrier frequency period. It is also true that six PWM outputs to the power device gate drive will occur at exactly one clock moment of the system clock at the beginning of the SYNC event. If synchronization is not implemented and the 8051 application software writes multiple data items to the MCE via the shared RAM, it is possible that some of the data are written in the previous MCE scan period while the rest of data are written in the current MCE scan period. Therefore, 8051 application software should use the SYNC signal for synchronization to insure that multiple data items are updated or read coherently within a particular scan period.

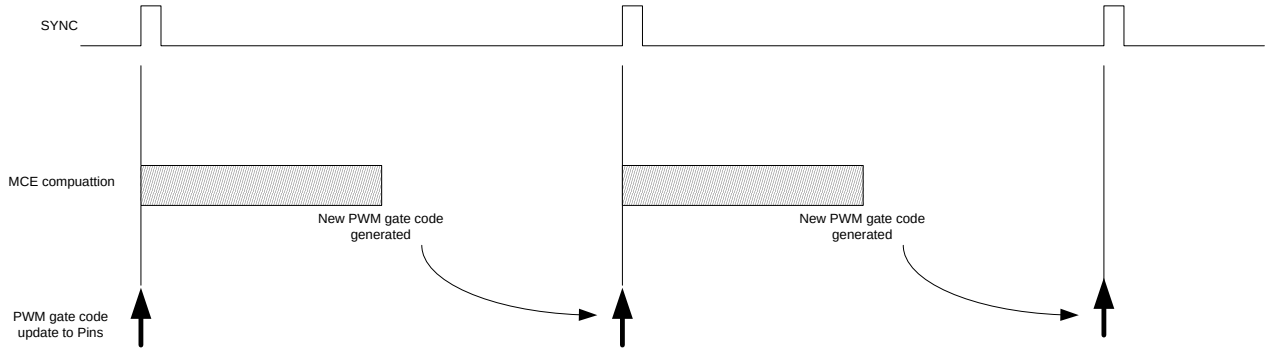


Figure 7. Timing of Sync and MCE Computation

For the IRMCx143, where the motor and PFC can be set to different PWM frequencies, the SYNC interrupt occurs only on the global SYNC pulse (the slower of the two, Motor PWM frequencies). For example, if the PFC is configured for a PWM frequency of 18 KHz and the motor is configured for a PWM frequency of 6 KHz, the SYNC interrupt will occur at a frequency of 6 KHz.

2 Setting Up the 8051 Development Tools

This section explains how to get started with 8051 application software development using the FS2 debug pod, Keil uVision tools, and the IRMCx100 Series IC.

2.1 Software Setup

2.1.1 Keil uVision Setup

In Keil uVision, the project options must be set appropriately for use with the IRMCx100 series IC and the FS2 debug pod. To configure the FS2 driver, display the Options window for your project and perform the following steps:

1. Select the Debug tab.
2. Click the radio button "Use:" and set the field to its right to *Fs2/Keil ISA-M8051EW Driver*, as shown in Figure 8 below. If this option does not appear in the list, the FS2 software has not been installed properly.

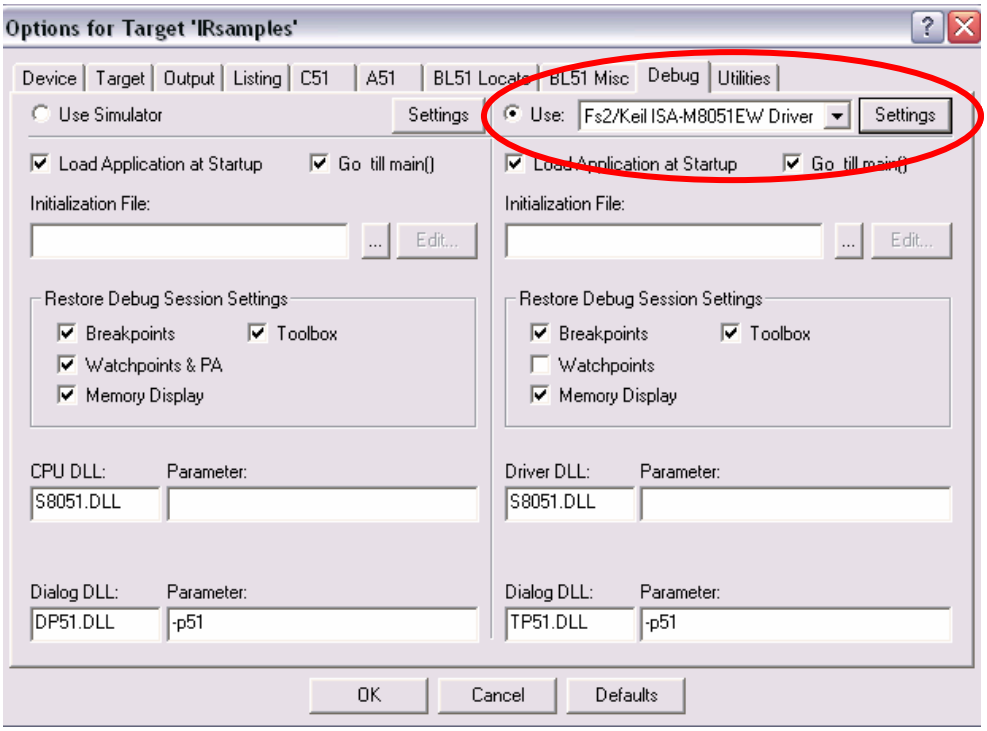
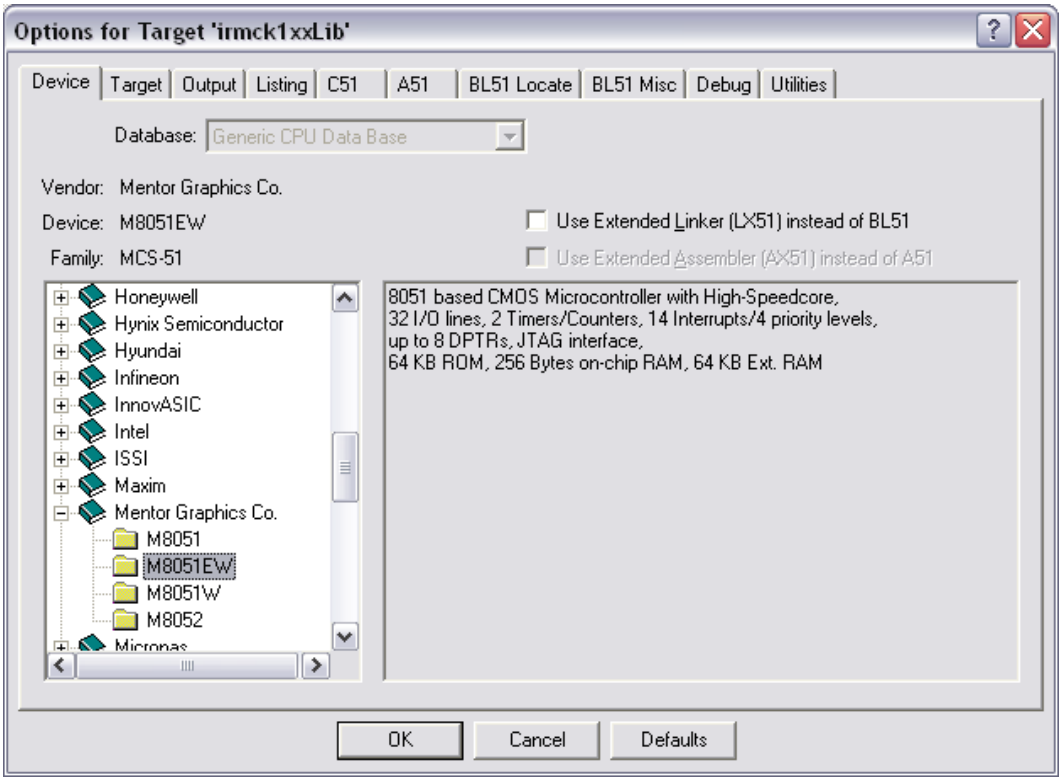


Figure 8. Debug Options Window for a uVision Project

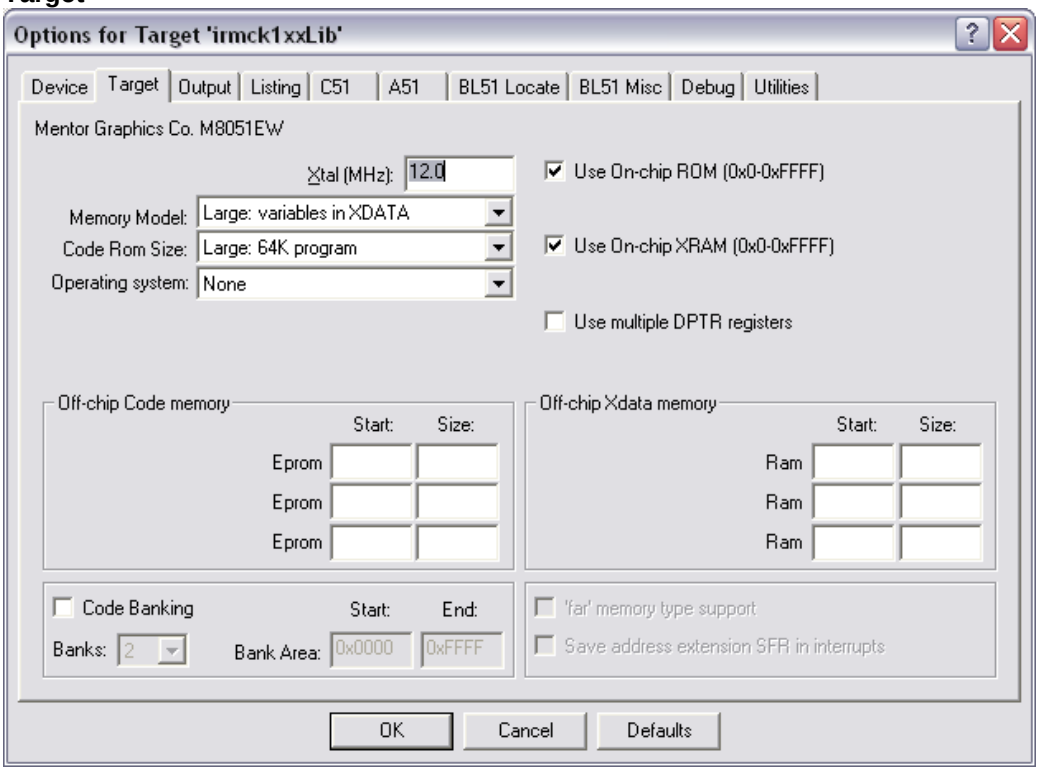
3. Select the BL51 Locate tab. The 8051 program (code) address range and data RAM (Xdata) address range must be defined properly. For example, the MCEDesigner Agent sets the following address ranges:
   Code Range: 0x0004-0x4F00, 0x5000-0x5FFF
   Xdata Range: 0x8A00-0x8FFF
   (The MCEDesigner Agent does not define the code range 0x4F00 – 0x4FFF, reserving it for the MCE Info structure defined in Appendix 2.)
4. When all options have been set as required, click OK in the Options window.

A complete display of the required uVision project options is provided below.

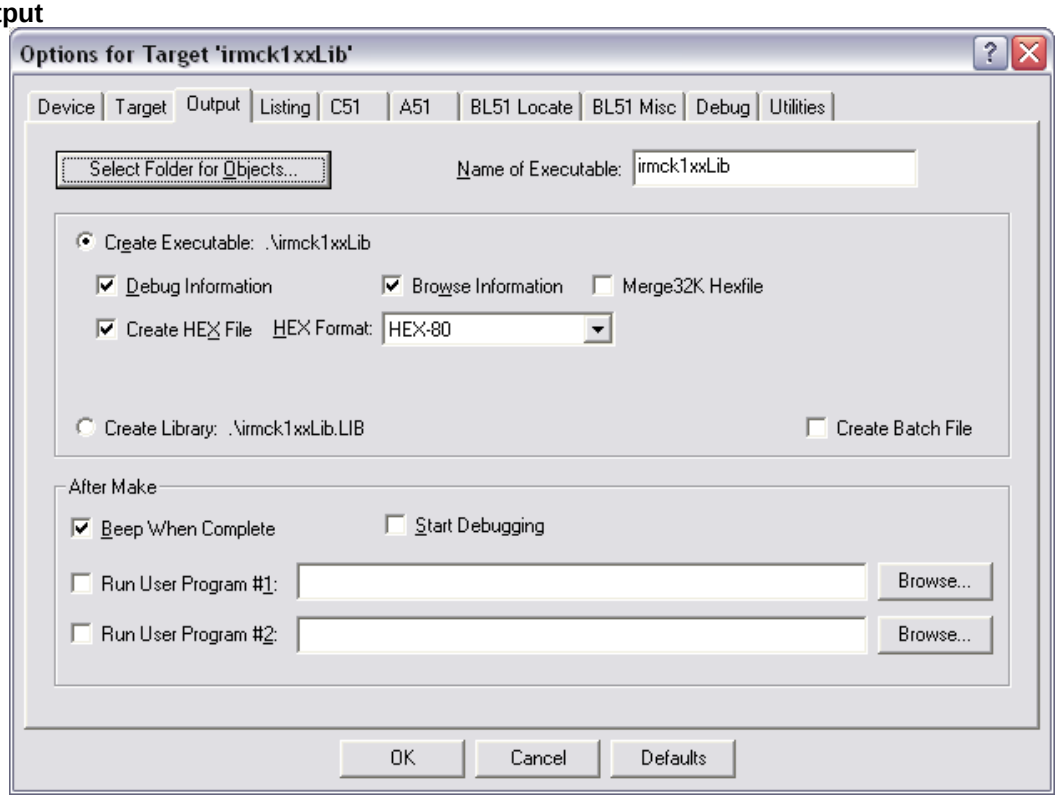
Device



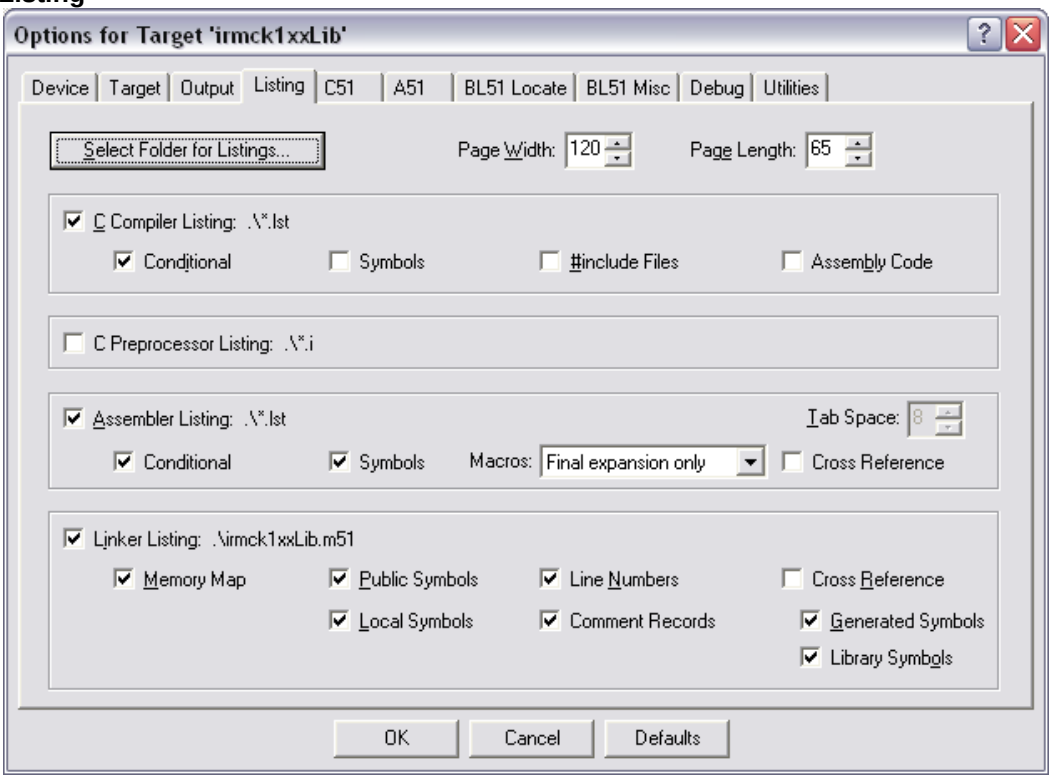
Target



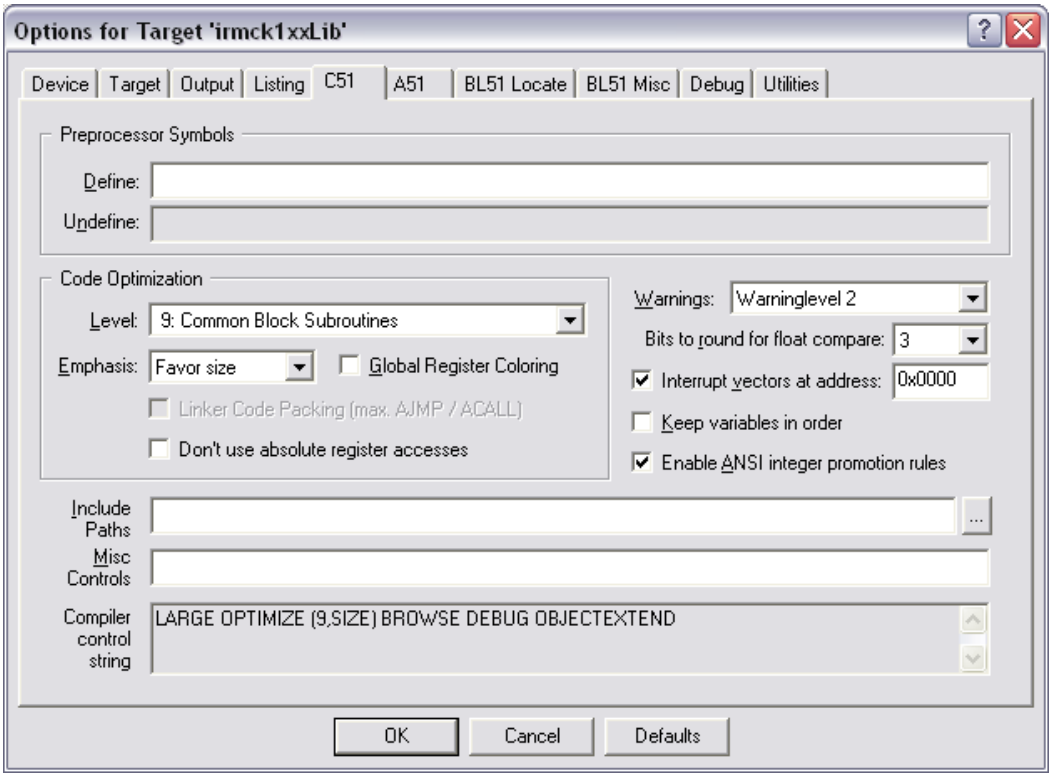
Output



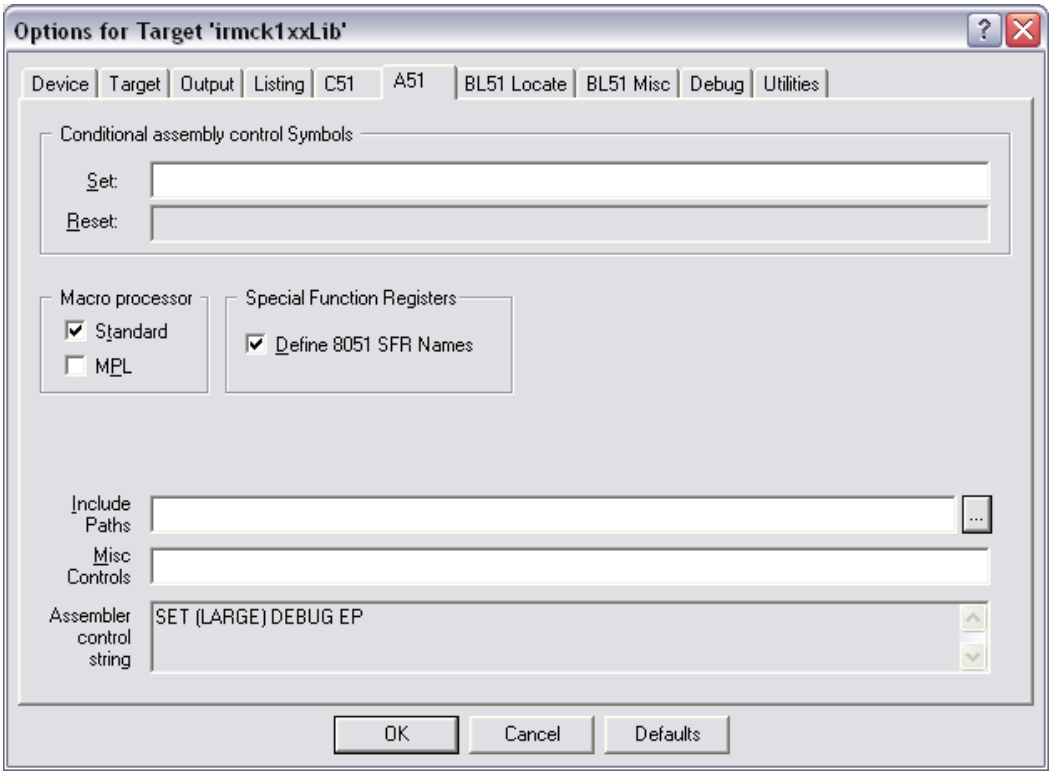
Listing


Options for Target 'irmck1xxLib'
Device
Target
Output
Listing
C51
A51
BL51 Locate
BL51 Misc
Debug
Utilities
Select Folder for Listings...
Page Width: 120
Page Length: 65
C Compiler Listing: .*.lst
Conditional
Symbols
#include Files
Assembly Code
C Preprocessor Listing: .*.i
Assembler Listing: .*.lst
Tab Space: 8
Conditional
Symbols
Macros: Final expansion only
Cross Reference
Linker Listing: .\irmck1xxLib.m51
Memory Map
Public Symbols
Line Numbers
Cross Reference
Local Symbols
Comment Records
Generated Symbols
Library Symbols
OK
Cancel
Defaults

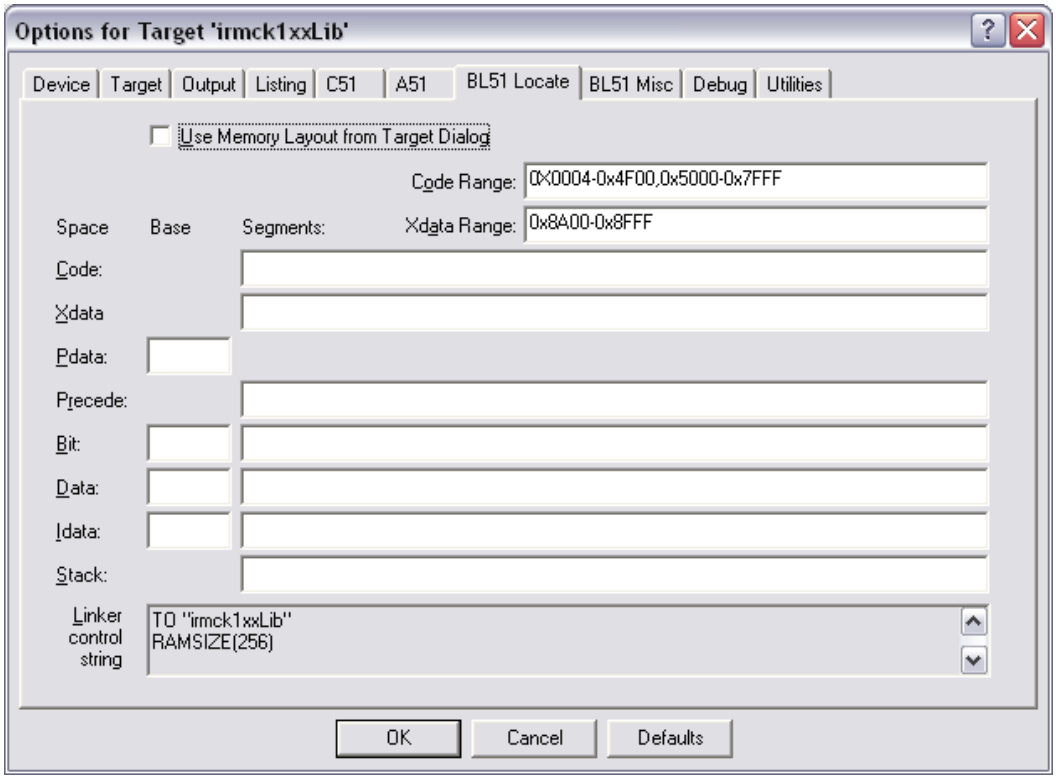
C51


Options for Target 'irmck1xxLib'
Device
Target
Output
Listing
C51
A51
BL51 Locate
BL51 Misc
Debug
Utilities
Preprocessor Symbols
Define:
Undefine:
Code Optimization
Level: 9: Common Block Subroutines
Emphasis: Favor size
Global Register Coloring
Linker Code Packing (max. AJMP / ACALL)
Don't use absolute register accesses
Warnings: Warninglevel 2
Bits to round for float compare: 3
Interrupt vectors at address: 0x0000
Keep variables in order
Enable ANSI integer promotion rules
Include Paths
Misc Controls
Compiler control string
LARGE OPTIMIZE (9,SIZE) BROWSE DEBUG OBJECTEXTEND
OK
Cancel
Defaults

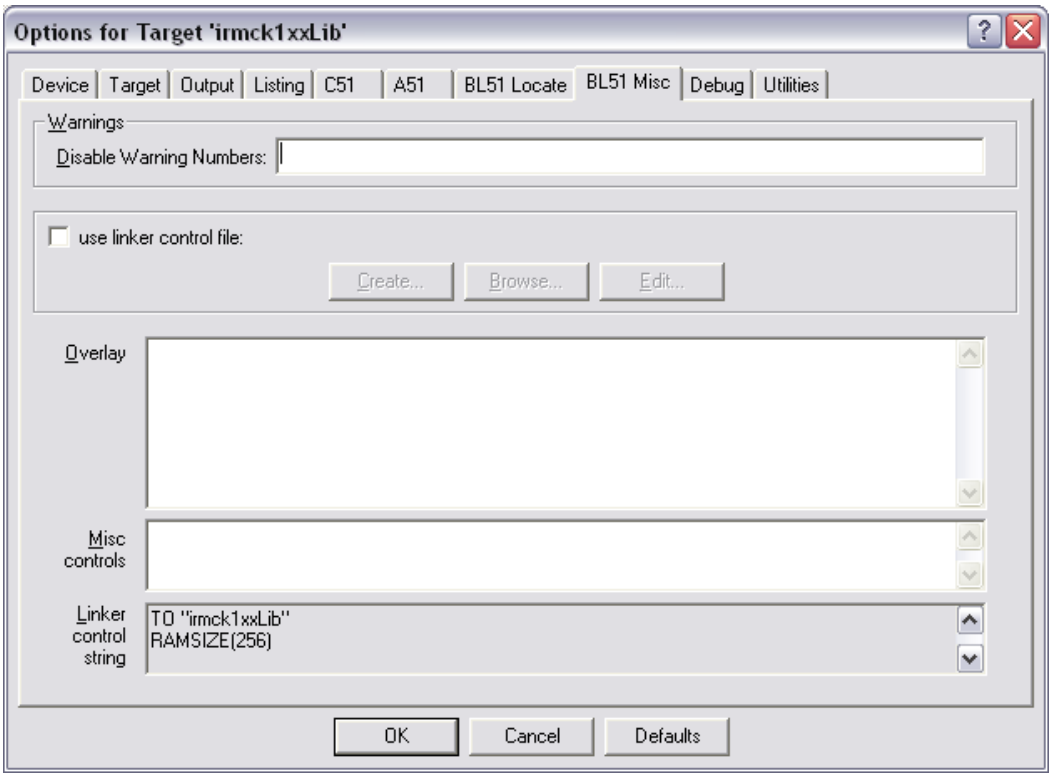
A51



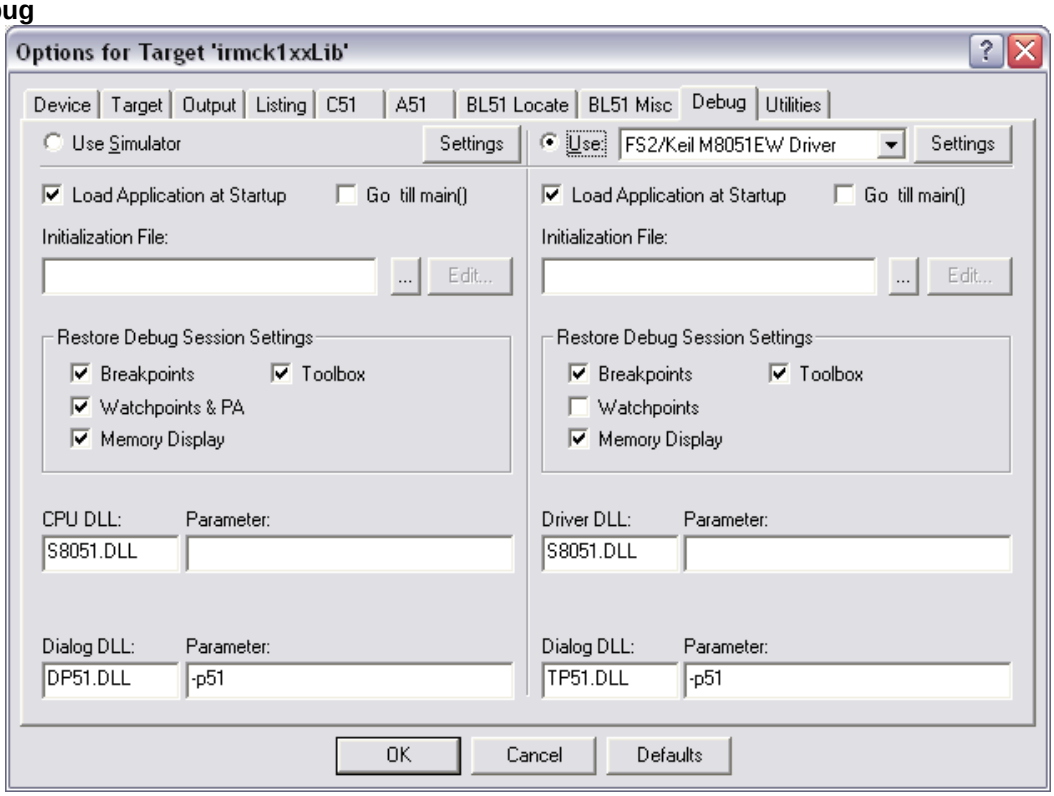
BL51 Locate



BL51 Misc

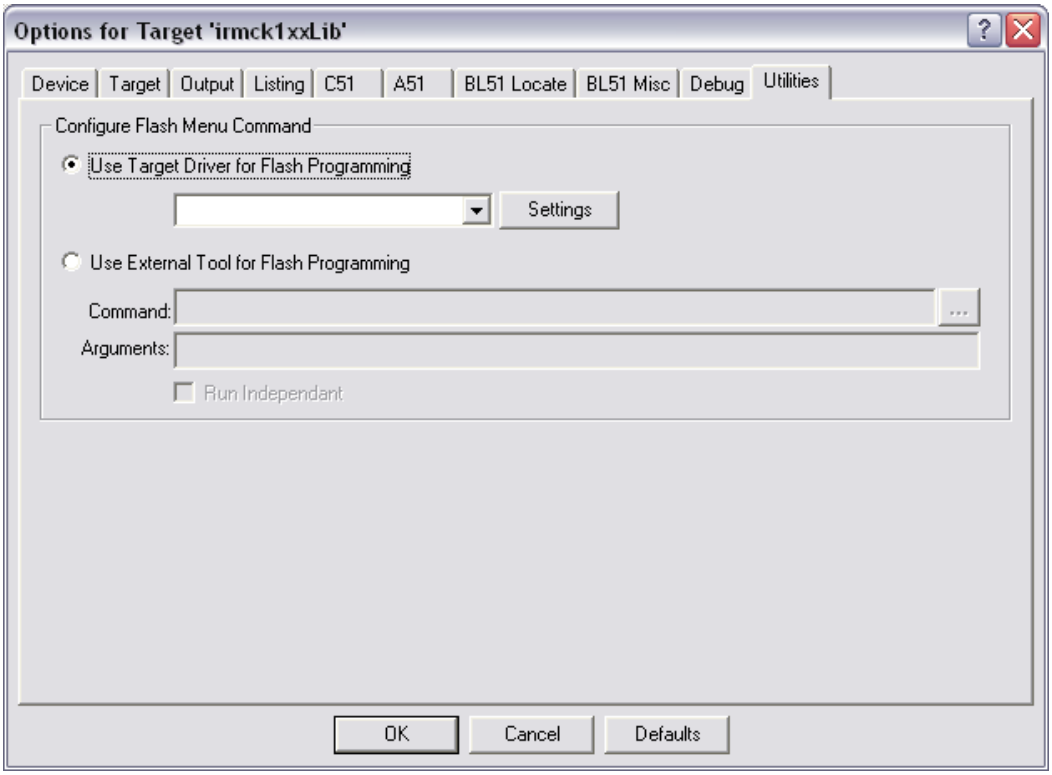


Debug



Note: The selection in the "Use:" box in the upper right of the Debug tab may not match this image's text exactly. **Be sure that the "Use" box is checked and the selection says "Fs2" and "Driver."** Also, be sure to uncheck "Go till main()."

Utilities



2.1.2 FS2 USB-based Debugger

The JTAG debugger facilitates debug of the 8051 program. It connects to a PC by a standard USB connector and to the board by a 10 pin connector provided in the FS2 kit. Some reference boards provided by IR may support the older parallel port-based debugger, which is not described here.

FS2 Configuration

Before any debugging can start the FS2 pod must be configured properly. The following setup should be used if the 10-pin JTAG header is used without isolation box IRMCS-ISO-V3:

1) ResetAsserted = 'low'
2) ResetNegated = 'high'
3) Tck Rate = 500000 (500kHz) ~ 5000000 (5MHz). 500kHz ~ 1MHz is recommended.

If isolation box IRMCS-ISO-V3 or the old type of 20-pin JTAG header is used, then set up as follows:

1) ResetAsserted = 'high'
2) ResetNegated = 'low'
3) Tck Rate = 500000 (500kHz) ~ 5000000 (5MHz). 500kHz ~ 1MHz is recommended.

Please note that setup of ResetAsserted and ResetNegated depend on whether there is a NOT logical on the RESET signal path to FS2. For example, when using IRMCS-ISO-V3 or hardware with a similar RESET circuit (IRMCS1043), set ResetAsserted to 'high' and ResetNegated to 'low'.

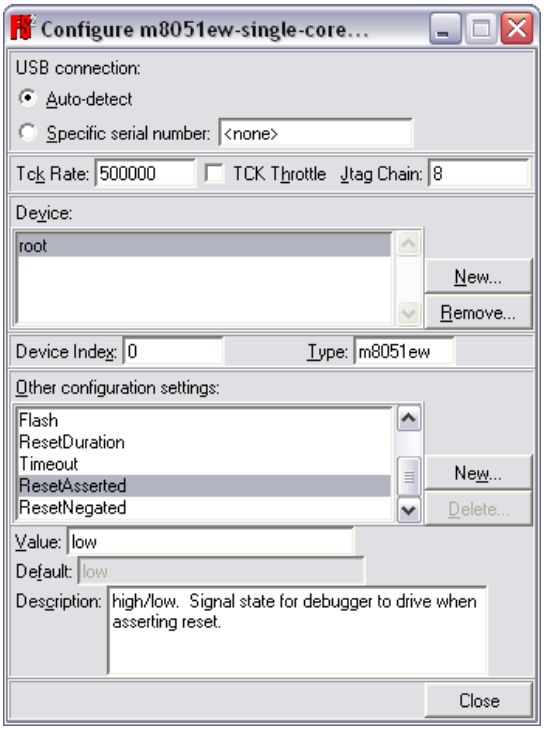


Figure 9. FS2 Configuration Dialog

These settings can be found in the 'Configure' window of the FS2 System Navigator Console program. They ensure that the proper reset logic levels are used. All other options should *not* be modified, as their default values are correct for this application.
When using Keil be sure to use the new driver included on the USB FS2 CD and not the older parallel port driver. The 'FS2 Getting Started Guide' covers exactly what to do. Be sure to install Keil first, then FS2, so that the FS2 installation configures Keil to correctly control the FS2 hardware.

2.2 Hardware Setup

Connect the UART interface to the computer where you have installed Keil uVision and 100SeriesKitMCEInstallerV2.1.exe or higher version software. On the IRMCS1043, this is achieved using an RS-232 (serial) cable from the control board to the PC. On the IRMCS1271, use connector J4 to IRMCS-ISO V3 isolation board. Also, connect the FS2 debug pod to the control board. On the IRMCS1043, the FS2 Pod interfaces to the control IC through connector J12. On the IRMCS1271, the FS2 Pod interfaces to the control IC through connector J3 to the IRMCS-ISO.

If the IRMCS1043 Reference Board is not used, please use IRMCS-ISO V3 isolation board or similar isolation circuit for safety and heed the warning below.

Warning!
When connecting the FS2 debug pod to the circuit board, the FS2 hardware can be damaged by the high voltage on the board if appropriate isolation is not used. The problem arises because the DC bus minus (GND) is not at the same potential as earth (or wall) ground. For this reason, if proper isolation is not used, it is recommended that the board be powered by a DC power supply with isolated ground when using the FS2 hardware.

2.3 Hardware and Software Startup

To properly start up the board and software for a debug session, follow these steps:

1. Apply power to the controller board to reset the controller.
2. Then power up the FS2 pod.
3. Start Keil uVision. Choose Project→Open Project and open the project file.
4. Choose Debug→Start/Stop Debug Session. The FS2 Console should come up briefly and a status bar in the lower left corner will display the progress in loading the 8051 code to RAM, shown in Figure 10 below.

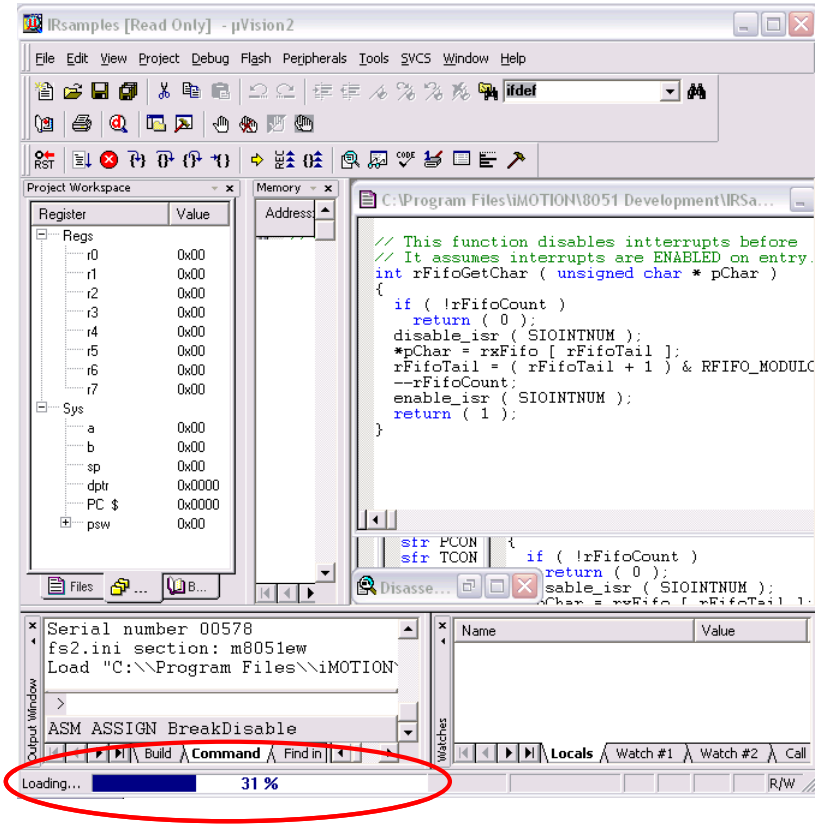


Figure 10. uVision Window During 8051 Program Load

5. Choose Debug→Go.
6. To exit from Keil after testing, choose Debug→Stop. Then Debug→ Start/Stop Debug Session. The program may be terminated at this time.

Note that Keil uVision does not actually download the 8051 application software to the IC when a debug session is started. The software to be debugged must be first programmed into OTP or flash memory using the MCEProgrammer utility or other method, as described in Section 4. Once a debug session is started, you can set breakpoints and run, and you can step through your program if interrupts are disabled. **However, *if you make changes to your software, you must program the new software to OTP or flash before beginning a new debug session*.**

If the software in memory does not exactly match the version most recently built in uVision, the debug session will behave unpredictably.

3 Sample Code

The sample code included with the Reference Design Kit, IRSamples, is intended to be a simple, easy-to-understand example. This program is not an efficient implementation with respect to function timing and optimization of processor usage during wait times.

The 8051 code can be used to implement more complex motor control functions than those described in Appendix 1 below. For example, a washing machine "wash cycle" requires that the motor accelerate rapidly in one direction, stop, and accelerate rapidly in the other direction. This could be implemented in such a way that the number of rotations is dependent on the whether the soil setting is low, medium, or high. Other operations that can be implemented include and auto-rebalance or PFC sequencing.

Note that a command interface other than the UART may be used. For example, to receive commands from the digital I/O pins, IRSamples could be modified so that it monitors the state of the SFRs corresponding to the appropriate pins and performs motor control functions as needed.

Once the designer has set certain variables, such as the clock rate, these variables could be stored in a configuration area of external EEPROM.

3.1 Sample Code Structure

The main functions that an embedded 8051 application performs are to configure the MCE with the proper drive parameters and to start, stop, and regulate the speed of the motor. These are the same functions performed by MCEDesigner; the difference is that the "intelligence" is transferred from the host application (MCEDesigner) to the embedded 8051 application. Any MCEDesigner function (a pre-defined series of register operations), including timing delays, can be implemented directly in the 8051 application so that it can control the motor independently or with simple external commands.

The 8051 controls and monitors the MCE by reading and writing interface registers. The registers are described in Section 3.1.2. The general steps that the embedded 8051 application must perform are:

1. Initialize the hardware—set clock frequency, initialize counters and timers, etc.
2. Start the MCE—verify that a valid MCE program has been loaded and initialize the MCE program counter.
3. Configure the drive—write drive parameters to MCE registers for the desired motor, and/or PFC operation.
4. Start and stop motor or PFC, change direction, change speed—keep track of current state in order to correctly implement command (e.g. don't change direction if motor is running).
5. Monitor for faults—periodic interrupts handle external commands, reset the Watchdog timer, check for faults/errors and shut down the drive if necessary.

For reference design kits based on the IRMCx143, IRSamples includes support for PFC. Certain functions described below are only valid for IRMCx143 and are noted as such.

3.1.1 Clock Frequency

In the function *main*, the MCE code must set up the phase-locked loop to generate the SYSCLK and 8051 clock frequencies by writing to the PLL SFRs. The clock frequency setup for IRMCK1xx and IRMCF1xx ICs is different, so the code first reads the register HWREV to determine which type of part it is. Next, the sample code can set the registers to configure 80, 100, or 123 MHz for the SYSCLK frequency and 20, 25, or 30.75 MHz for the 8051 clock frequency depending on the compiler directive #ifdef. The clock frequency is defined in the header file, motorctrl.h. Refer to Section 1.4 for more information on clock rate configuration and

recommended clock frequencies. For detailed definitions of the PLL special function registers, see the IRMCx100 Reference Manual.

Based on the configured clock frequencies, the code sets the proper UART baud rate, I^2C baud rate and timer initialization values, also selected by #ifdef. (UART and I^2C baud rates are based on SYSCLK, while the timer counter value is based on the 8051 clock rate.) Some drive parameters (such as PwmPeriod, which sets the motor PWM frequency) are also configured based on the clock frequency. The user should generate new drive parameters using the MCEWizard tool if a different clock rate is configured.

Note: The PLL registers are the only part of IRSamples which are dependent on the type of part, IRMCK100 or IRMCF100.

3.1.2 Registers

There are several types of registers, listed and described below.

1. Special Function Registers (SFRs)—SFRs can only be accessed by the 8051 microprocessor. Only a subset of the SFRs is described in this Guide. A complete list and description of the SFRs can be found in the IRMCx100 Reference Manual. The SFRs can be used to:
    - Initialize and modify processor registers
    - Configure and read I/O ports
    - Set the clock frequency
    - Configure, initialize and reset general-purpose timers
    - Enable analog features such as op-amps
    - Enter and exit low-power modes
    - Configure and enable interrupts
    - Access shared RAM using the coherent read/write mechanism

    Since SFRs can only be accessed by the 8051 microprocessor, synchronization between processors is not an issue and writing to these registers is relatively simple. They are defined in *irmcx1xx.h* using a special “sfr” keyword. Each SFR is assigned a name that corresponds to its memory address and the register can be written and read using the name, as with any other variable. Below is an example showing clock frequency setup through the PLL0 – PLL3 SFRs.

    ```
    PLLF0 = 0x80;
    PLLF1 = 0xAC;
    PLLF2 = 0x62;
    PLLF3 = 0x52;
    ```

2. Peripheral Registers—These registers are accessible to both the MCE and 8051 microprocessors, but are only used by the 8051 application. These registers have functions similar to the SFRs described above, and can be used to:
    - Configure, initialize and reset special-purpose timers
    - Configure the UART
    - Configure and use I^2C/SPI serial interface

    Since peripheral registers are only accessed by the 8051 microprocessor, synchronization between processors is not an issue and writing to these registers is relatively simple. They are defined in *PeripheralRegs.c* as fixed-address variables so that they can be directly read and written. Below is an example showing service of the watchdog timer by reading the watchdog limit register *WDLIML*.

    ```
    watch = WDLIML;
    ```

3. Fixed Motion Hardware and Motion Firmware Registers (MHRs and MFRs)—These registers are accessible to both the MCE and 8051 microprocessors. The 8051 microprocessor accesses the registers through shared memory using a special coherent read/write mechanism that ensures proper synchronization with the MCE. Most MHRs and MFRs only need to be configured once before running the motor, which can be done by the 8051 application (see the "configure" function in IRSamples). Additionally, the 8051 application may write to certain MHRs and MFRs to start and stop the motor or to adjust parameters due to varying operating conditions. The MCE may modify a subset of the MHRs and MFRs every PWM cycle.

   MHRs and MFRs used in IRSamples are defined in the file MotionRegs.h. Each register name begins with the characters "reg_" and is defined with a value that corresponds to its 8051 memory address. The functions *DoRegRd* and *DoRegWr* in the sample code are provided to read and write these registers using the coherent read/write mechanism implemented in the assembly-language functions doCoherentRd and doCoherentWr. It is recommended that the coherent read/write functions be used without modification as the specific sequence of operations is critical for correct operation. Below is an example of writing the value "4" to MFR MtrSeqCtrl.

```
DoRegWr ( reg_MtrSeqCtrl, 4 );
```

4. User-defined MCE Registers—These are registers defined in the MCE Simulink design. In contrast to MHRs and MFRs, MCE registers do not have fixed memory addresses. These registers are assigned RAM addresses by the MCE Compiler. The compiler outputs a header file that must be incorporated into the 8051 code (detailed in Section 3.2.2). The names assigned by the MCE compiler to the MCE registers have the format:

   <Simulink Sub-system>_<register name in Simulink>

   MCE registers are accessible to both the MCE and 8051 microprocessors. The 8051 microprocessor accesses the registers through shared memory using the same coherent read/write mechanism used for MHRs and MFRs. The same functions, *DoRegRd* and *DoRegWr,* are used. Below is an example of setting the target speed (of the Motor1 sub-system) to the value 6000.

```
DoRegWr (Motor1_TargetSpeed, 6000 );
```

3.1.3 Files and Functions of Sample Code

The sample code is composed of several C source files, which divide the functions into groups according to their use. The C and assembly source files are:

1. *main.c* — Execution begins here. The *main* function calls each of the samples. The last sample function, *MotorCtrl*, does not return.

2. *regIf.c* — This file contains functions to read and write registers in shared RAM (MHRs, MFRs and MCE registers) with guaranteed coherency using 8051 SFRs. See *Coherent.SRC* for the implementation of the coherent read/write interface. Examples of calls to the register interface functions can be found in *MotorCtrl.c*.

3. *EepromI2C.c* — This file contains sample code to read and write the EEPROM using the I^2C interface.

4. *Timer.c* — This file contains a function that initializes timer 1 to generate interrupts at 2 msec intervals. A global variable "systicks" is incremented on each interrupt and the FaultFlags register is checked for a fault condition. The interrupt service routine also

services the Watchdog timer. The Watchdog timer must be serviced periodically; otherwise the IC is automatically reset. Timer 1 setup is based on the 8051 clock rate (see Section Appendix 1).

5. *MceBoot.c* — This file contains functions to initialize the MCE using code that has been programmed to OTP memory by the MCEProgrammer tool. It assumes that the automatic boot process has copied the MCE code from non-volatile memory to shared RAM and an "MCE Info" structure from the memory to a fixed location in 8051 program RAM. See Appendix 2 for the description and function of the MCE Info structure.

   The function *StartMce* first copies the MCE Info structure from 8051 program RAM to a location in data RAM and verifies the validation field in the structure. If the validation field is incorrect, the entire structure is assumed to be invalid and the MCE is not initialized. Otherwise, the MCE Info structure provides the starting load address in RAM and the MCE execution address. The *StartMce* function uses this information to zero the MCE data area. The function *doMceBoot* is called to initialize the MCE special registers and begin MCE execution.

6. *asyncDriver.c* — This file contains functions to set up the UART and read and write data using FIFO (first-in-first-out) buffers. The following functions are included in the file:

   sioIsr - This is the UART interrupt service routine, which handles transmit and receive interrupts. Received characters are placed in the receive FIFO. Characters to be transmitted are taken from the transmit FIFO.

   sioInit - This function initializes the transmit and receive data structures and the peripheral registers that control the UART.

   flushTx - Initializes the transmit FIFO.

   flushRx - Initializes the receive FIFO.

   setBaudRate - Inializes the UART baud rate register for 57,600 bps, based on the system clock rate (see Section Appendix 1).

   putChar_ - This function is called from a higher level (such as the *MotorCtrl* function) to transmit a character. If the transmitter is currently busy, it adds the character to the transmit FIFO. If no transmission is already in progress, it writes the character directly to the UART transmit buffer. The function returns 0 if the transmit FIFO is full (character cannot be accepted for transmission); or 1 if successful.

   getChar_ - This function is called from a higher level to read a received character from the receive FIFO. It returns 0 if the receive FIFO is empty (no character available) or 1 if successful.

   xFifoRoom - Called from *putChar_* to check the status of the transmit FIFO. Returns 0 if the transmit FIFO is full; 1 otherwise.

   xFifoPutChar - Called from *putChar_* to add a character to the transmit FIFO. Returns 0 if the transmit FIFO is full; 1 if the character was successfully added to the FIFO.

   xFifoGetChar - Called from *sioIsr* to get the oldest character from the transmit FIFO. Returns 0 if the transmit FIFO is empty; 1 if a character is removed from the FIFO.

   rFifoRoom - Called from *sioIsr* to check the status of the receive FIFO. Returns 0 if the FIFO is full; 1 if the received character was successfully added to the FIFO.

rFifoPutChar - Called from *sioIsr* to add a character to the receive FIFO. Returns 0 if the receive FIFO is full; 1 if the character was successfully added to the FIFO.

rFifoGetChar - Called from *getChar_* to get the oldest character from the receive FIFO. Returns 0 if the receive FIFO is empty; 1 if a character is removed from the FIFO.

IMPORTANT NOTE: The transmit and receive FIFOs are manipulated from both the interrupt level and the "task" (non-interrupt) level. For this reason, it is very important to ensure that UART interrupts are disabled while characters are added to and removed from the FIFOs at the task level.

7. *MotorCtrl.c* — This file contains a simple example of motor drive configuration and control. It reads character commands from the serial port using the functions provided by the UART driver. You can use a HyperTerminal (or equivalent) connection to send commands and read responses. A list of supported commands and their descriptions can be found in Section 3.2.3.

   The function *MotorCtrl* checks that the MCE versions defined in regIf.c and loaded from non-volatile memory match before allowing motor control operations. The version number defined in regIf.c comes from the header file generated by the MCE Compiler. The version number in non-volatile memory is stored there by MCEProgrammer as part of the MCE Info structure .

8. *PeripheralRegs.c* — Fixed-address variable definitions for the peripheral registers.

9. *Coherent.SRC* — Assembly-language functions to read and write shared RAM registers using SFR registers for coherent data transfer.

10. *utils.c* — Utility functions to enable and disable a particular interrupt, identified by the interrupt number, as defined at the beginning of the file.

3.2 Running the Motor

To run the sample code, send commands to the controller and read its responses by connecting a serial cable between the target platform and your PC. On the PC, run a “terminal emulation” program, such as HyperTerminal or Tera Term Pro. Set up the terminal emulation program to use the same COM port you use with MCEDesigner and configure the settings as follows:

- 57,600 bps
- 8 bits
- no parity
- 1 stop bit
- no flow control

3.2.1 Drive Configuration

After power-up, the motor and PFC will not run properly until the MCE has been configured with the correct parameter settings. These configuration parameter values are generated using the MCEWizard tool. The scaled values produced by MCEWizard should replace the sample values defined in *parameters.h*. After clicking “Calculate” on MCEWizard’s “Verify Parameters” page, click “Export to a C header file (.h)” to generate a new set of parameter definitions for the *parameters.h* file.

3.2.2 MCE Header File

The binary (.bin) file containing the MCE program is generated when the Simulink model file is compiled. The compiler can also produce a C header file (.h) which contains:

- definitions of user-defined MCE registers;
- register map structures for addressing of the user-defined MCE registers; and
- product and version identification.

The C code from the generated header file should be used to replace the sample code at the beginning of *regIf.c* and *regIf.h* as described below.

In *regIf.c*, replace the section titled "COMPILER GENERATED INITIALIZATIONS" with the corresponding section from the header file generated by the MCE compiler. Specifically, replace the following sections in *regIf.c*:

```
char ir_productID = 74;
char ir_designID [] = "IRMCS1271_Release_1_0";
char ir_vers [] = "1.436";

RegMapType RegMap [] = {
  { 2, 0, 0, 16 }, /* 0 */
  { 3, 34, 0, 16 }, /* 1 */
    ...
    ...
  { 3, 2, 0, 16 }, /* 17 */
  { 3, 0, 0, 16 }, /* 18 */
};

unsigned short PageBase [] = {
  0x8000,
   ...
   ...
  0x0000,
};
```

Similarly, in *regIf.h*, replace the section, reproduced below, titled "COMPILER GENERATED DEFINITIONS" with the corresponding code from the MCE Compiler's header file.

```
/* Register map array indexes */
#define Motor1_MotorSpeed     0
#define Motor1_AccelRate      1
    ...
#define Motor1_TargetSpeed    17
#define Motor1_VhzEnable      18

/* Definitions for Page field in RegMap and indexes into PageBase
*/
#define PAGE0_RD     0
    ...
#define PAGE3_WR     7

/* Register map structure definition */
typedef struct {
   unsigned char Page;
   unsigned char ByteOffset;
   unsigned char BitOffset;
   unsigned char BitLength;
```

```
} RegMapType;
```

Note that the MCE design ID and version number of the header file must match that of the MCE code loaded from OTP memory for correct operation. If the Simulink model is changed, then a new header file must be created during compilation and added to the code as described in the paragraph above.

As shipped, the sample code is configured to correctly write to the MCE registers defined in the Reference Design Kit and has the proper drive parameters to run the Golden Age GK6040-6AC31-WE motor.

3.2.3 Motor and PFC Functions

The sample code treats the motor as a state machine, with three states: DRIVE_IDLE, DRIVE_RUN and DRIVE_FAULT. The function *MotorCtrl* takes input commands from the serial port and passes valid commands to function *MotorSeq*. Based on the current motor state, *MotorSeq* calls appropriate functions to implement the command or returns an error indicating that the command was invalid. If an invalid command is entered, 'Invalid Command' is returned to the HyperTerminal display. Listed below are the commands supported from the function *MotorCtrl*, with explanations of their operation.

C or c
Configure motor drive and clear faults. 'Configured' will be echoed back on the UART if successful. If the motor is running, the command is ignored and 'Invalid Command' is returned instead.

+
Set forward direction. 'Forward' is echoed when the operation is complete. If the motor is running or in a fault condition, the command is ignored and 'Invalid Command' is sent instead.

-
Set reverse direction. 'Reverse' is echoed when the operation is complete. If the motor is running or in a fault condition, the command is ignored and 'Invalid Command' is sent instead.

F or f
Clear fault condition. 'Fault Clear' is echoed when the operation is complete. If the drive is not in a fault condition, the command is ignored and 'Invalid Command' is sent instead.

G or g
Run motor. The motor is placed in run state and turns in the configured direction at a low speed. 'Started' is echoed when the operation is complete. If the motor is already running or in a fault condition, the command is ignored and 'Invalid Command' is sent instead.

S or s
Stop motor. The motor is stopped and 'Stopped' is echoed when the operation is complete. If the motor is already stopped or in a fault condition, the command is ignored and 'Invalid Command' is sent instead.

R or r
Set motor speed. The program prompts for exactly four decimal digits (0 - 9) defining the target speed in rotor RPM. If the motor is not running the command is ignored and 'Invalid Command' is echoed. If the requested speed is out of range for the motor (according to the value of "#define Mtr_Max_Speed") then the Mtr_Max_Speed value is used. Otherwise, the operation is performed after all four digits have been received, at which point 'Speed Set' is echoed. If a character other than a digit is received, an 'X' is echoed and the command is aborted.

?
(1) Get motor speed. This command returns the motor speed when the drive is running. The current speed is output in motor RPM. This RPM calculation relies on the parameters generated by MCEWizard.
(2) Read FaultFlags. When the drive is in a fault state, this command returns the value of the FaultFlags register. The register value is displayed in hexadecimal format.
If the drive is not running and there is no fault condition, the command is ignored and 'Invalid Command' is returned.

H or h
Catch-Spin Start. This begins the catch-spin startup sequence for the motor. In this mode, the MCE firmware automatically monitors the speed and direction of the motor to determine if the motor should be stopped and reversed, or if the motor is already going in the correct direction and catch it. This startup mode is suitable for an instance where the motor may already be in motion due to outside forces (such as wind blowing a fan). This command is allowable only in the idle state, otherwise 'Invalid Command' is echoed. At the end of the sequence, 'CatchSpin Complete' is echoed.

T or t
Ramp Stop. This function slowly ramps the motor down to zero speed (rather than simply stopping the motor by halting the PWM as the Stop Motor command does). Upon successful stopping of the motor 'Ramp Stop Complete' is echoed. The rate is determined by the *rampTime* variable, which is the time in seconds to ramp to zero.

Z or z
Zero Vector Brake. This function will turn on the zero vector brake command for 20 seconds, then halt the PWM and turn off zero vector brake. In the case of a fault, the function will break out of the 20 second wait time and halt the PWM.

>
<
These commands are used for direct register access. The > command writes to a register and < reads from a register. When one of these commands is given, the controller will prompt the user for the five-digit register offset in decimal format. For an MHR or MFR, use the 8051 address shown in the reference manual or defined in MotionRegs.h. For an MCE register, the register offset can be found in the .h or .map file output from the MCECompiler or in the .irc file. If the write command (>) is given, then the controller will prompt for a five-digit, decimal value to be written. These commands are valid regardless of the sequencer state.

P or p (IRMCx143 only)
PFC control. The program prompts for one digit '1' or '0' to control the PFC. Input '1' enables PFC, and input '0' disables PFC. This command is valid when the drive is idle or running. Please note that PFC only functions with IRMCx143. Before compiling the sample code, please check that the hardware is consistent with the definition in "MotorCtrl.h".

3.3 Extending Functionality

IRSamples only supports basic motor control and is configured to work only with the released version of the development kit MCE design. It is likely that new register names are defined in a new MCE implementation. If this is the case then modifications to the base IRSamples are required.

Before making changes to the IRSamples code for a modified MCE design, import the new motor parameters and the new header (*.h) file from the MCE Compiler.

If MCE register names are changed, it is necessary to not only update the *regif.h* and *regif.c* files as outlined previously, but also to modify all references to the old MCE register names in IRSamples. Most of these modifications are located in *MotorCtrl.c* (*MotorSeq* function) or *Timer.c* (*Timer1* function).

3.4 Troubleshooting

No characters are echoed in HyperTerminal (or other terminal emulator):

- If the FS2 debug pod is attached and powered on, then you must have a “debug session” started in the uVision debugger, and you must start execution of the 8051 application after download is complete.
- Check that the computer running the terminal emulation program is connected (by serial cable) to the hardware. Also, verify that program is using the correct COM port with the correct communication options as described in Section 3.2. The COM port should be the same one that MCEDesigner uses.
- The IRMCS-ISO-V3 isolation board has LEDs to indicate UART communication status (LED2 & LED3 for USB to RS232 communication and LED5 & LED7 for RS232 communication). When there is TXD or RXD data, the corresponding LED will blink. By observing the LED status when sending commands from the PC, you can tell which side is not receiving the expected UART data.

The Run Motor command (‘g’) is not accepted.

- Use the ‘?’ command to check for a fault condition.

The motor does not turn when commanded.

- If the FS2 debug pod is attached, it must not be powered on until after the target platform is powered on. If the FS2 pod is powered on first, it prevents the IRMCK1xx hardware from loading the MCE program from non-volatile memory into MCE program RAM at power up.
- Check that the correct drive parameters are entered into parameters.h. If you modified the drive parameters in MCEWizard, verify that the IRSamples project has been rebuilt to include the new parameters.h file.
- Check that the MCE is properly started—Set a break point in *MCEBoot.c* to see whether the *doMceBoot* function is called. If not, reprogram the IC and be sure to include the MceInfo structure at the default address of 0x4FA0.

4 Programming the Control IC

This section describes how to load the MCE program to RAM for testing and how to program the 8051 and MCE programs to non-volatile memory. Several options for programming are described below:

1. Use the IRMx100 Programming Board (IRMCx100 Programming Kit, IRMCK171PROG board, or IRMCK143 PROG board), together with the IRMCx Download Cable (IRCable) or FS2 to program the IC.
2. Use on-board programming with one of the IRMCS1xxx 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.

Because the IRMCF1xx parts contain flash memory and the IRMCK1xx parts contain OTP memory, the non-volatile programming procedures differ somewhat for the two device types. Flash programming is described in Section 4.1 and OTP programming is described in Section 1.1. Section 4.6 describes the RAM download procedure, which is the same for all IRMCx100 parts. Custom programming methods are covered in Appendix 1.

As input, MCEProgrammer imports a design's MCE program (.bin file) and 8051 program (.hex file), converts the program files into a memory image in the proper format for storage in flash or OTP memory (see Section 1.5.2 for more information) and then programs the memory image to flash or OTP.

4.1 Programming Flash Memory

This section describes how to program flash memory on the IRMCF1xx ICs. Flash memory is programmed while the IC is installed in the reference design board using the MCEProgrammer utility. For more details about using MCEProgrammer, refer to the MCEProgrammer User's Guide.

MCEProgrammer supports two methods of programming flash memory:

- Flash can be programmed by communicating with the IRMCx Download Cable (IRCable).
- Flash can be programmed by communicating with the FS2 JTAG debugger (Console program).

For either programming method, MCEProgrammer always formats a complete binary image of the entire flash memory, which includes the components listed in the table below. The entire flash memory is erased and reprogrammed.

Component	Binary File	Address Range
8051 Program (MCEDesigner agent, IRSamples, or user's custom application)	User-specified .hex	0x0000 – 0xCBFF
MCE Program	User-specified .bin	0xCC00 – 0xFBFF

Note:
When programming IRMCF1xx flash on the reference design board, *do not* turn on the VPP switch. VPP is required for programming IRMCK1xx OTP memory only.

4.2 Programming OTP Memory

This section describes how to program OTP memory on the IRMCK1xx ICs via IRCable, MIPS/FS2 or Corelis (USB-1149.1/E or USB-1149.1/1E) JTAG device.

OTP memory is programmed using the MCEProgrammer utility, which can perform direct OTP programming by communicating with the IRMCx Download Cable (IRCable) the FS2 JTAG debugger (System Navigator Console program) or a Corelis (USB-1149.1/E or USB-1149.1/1E) JTAG device.

As input, MCEProgrammer imports a design's MCE program (.bin file) and 8051 program (.hex file), converts the program files into a memory image in the proper format for storage in OTP memory (see Section 1.5.1 for more information) and then programs the memory image to OTP.

To program the OTP, the user has several options:

1. Use the IRMCK100PROG Programming Board with either IRCable or Corelis. This option is described in Section 4.4.
2. Use the reference design board with IRCable or FS2 to program the IC on-board through the provided debug connector. This option is described in Section 4.5.
3. Design custom hardware to communicate with the IRMCK100 series IC through the JTAG pins. This option is described in Appendix 1.

The JTAG headers on the Programming Board can interface to IRCable and the Corelis JTAG device. For on-board programming IRCable and the FS2 interface are supported. From a user's point of view, the main difference among the three options is the programming speed – IRCable and Corelis are considerably faster than FS2. The main features for the programming options are listed in the table below.

Method	On-board support	IRMCK1xxPROG support	Programming Speed (32k)
IRCable	Yes	Yes	Approx. 12 sec total for program and verify
MIPS / FS2	Yes	No	Program. approx. 25 sec, Verify approx. 30 sec
Corelis	No	Yes	Approx. 7 sec total for program and verify

4.2.1 Programming Pins

OTP programming is performed through the standard JTAG interface available on the IRMCK100 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 IRMCS1271/1043 or IRMCS-ISO V3.0 isolation board). This voltage is supplied to the dual-purpose pin VPP/P1.5. When 6.75V is supplied to this pin it will act as the supply rail for the internal OTP memory. If performing programming in-circuit it is important to either verify that external components can withstand 6.75V or to isolate the VPP/P1.5 pin during programming. An easy way to isolate devices is to have a jumper present on the VPP line; after the OTP is programmed the jumper can be put in place.

Refer to Section 4.5.1 for the recommended procedure when programming OTP memory on a reference design board.

4.3 Using MCEProgrammer

This section gives an overview of using MCEProgrammer to program an IC. The procedure is similar regardless of the type of memory (flash or OTP), programming interface (IRCable, MIPS/FS2 or Corelis) or the programming platform (programming board or reference design board).

For a complete description of the MCEProgrammer tool, including all option settings and types of operations, please refer to the MCEProgrammer User's Guide.

4.3.1 Initial Setup for MIPS/FS2

The MIPS/FS2 interface requires the installation of a driver that is included with the MIPS System Navigator tools installation.

Before using MIPS/FS2 with the MCEProgrammer tool for the first time, you must set up the directory path of the FS2 Console program in MCEProgrammer. From the Tools menu, select "FS2 Setup" and enter the location of the FS2 Console program. If the FS2 software is installed in the default location, then this location will be as shown in Figure 11.

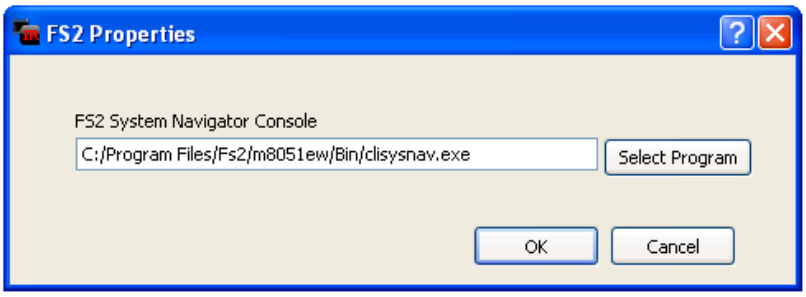


Figure 11. Setup of FS2 Console Location

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.

4.3.2 Initial Setup for IRCable

IRCable 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 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 12. Choose the COM port that corresponds to the CP2102 USB to UART device. Make sure the default baud rate (256,000 bps) is selected. Then click OK.

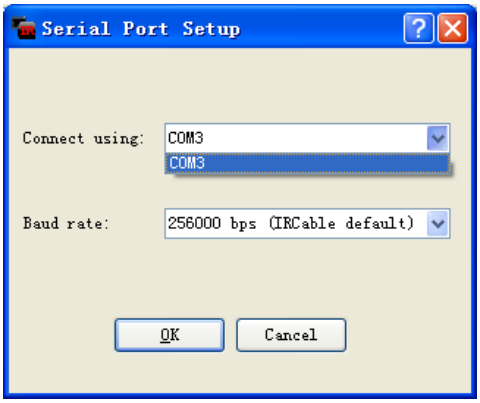


Figure 12. Serial Port Setup

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.

4.3.3 Initial Setup for Corelis

The Corelis USB-1149.1/E or USB-1149.1/1E JTAG device requires a driver that is included with the Corelis Scan Function Library installation.

4.3.4 Option Selection

All communication with the user is done through the graphical interface shown in Figure 13.

At the main window, make the following selections:

- Select the “Product” name from the pull-down list (e.g. IRMCF143).
- Select the “Operation”, “FS2 JTAG: Program flash” or “IRCable: Program flash”.
- Provide the locations of the 8051 .hex file and MCE .bin file you want to program.
- If you want to protect the contents of the flash from being read back, check the “Protect Flash” box.
- “MCE Info Structure Address” allows the user to change the location of the MCE Info structure, which is required for use with MCEDesigner (see Appendix 2). There is a default MCE Info address pre-defined for each product.

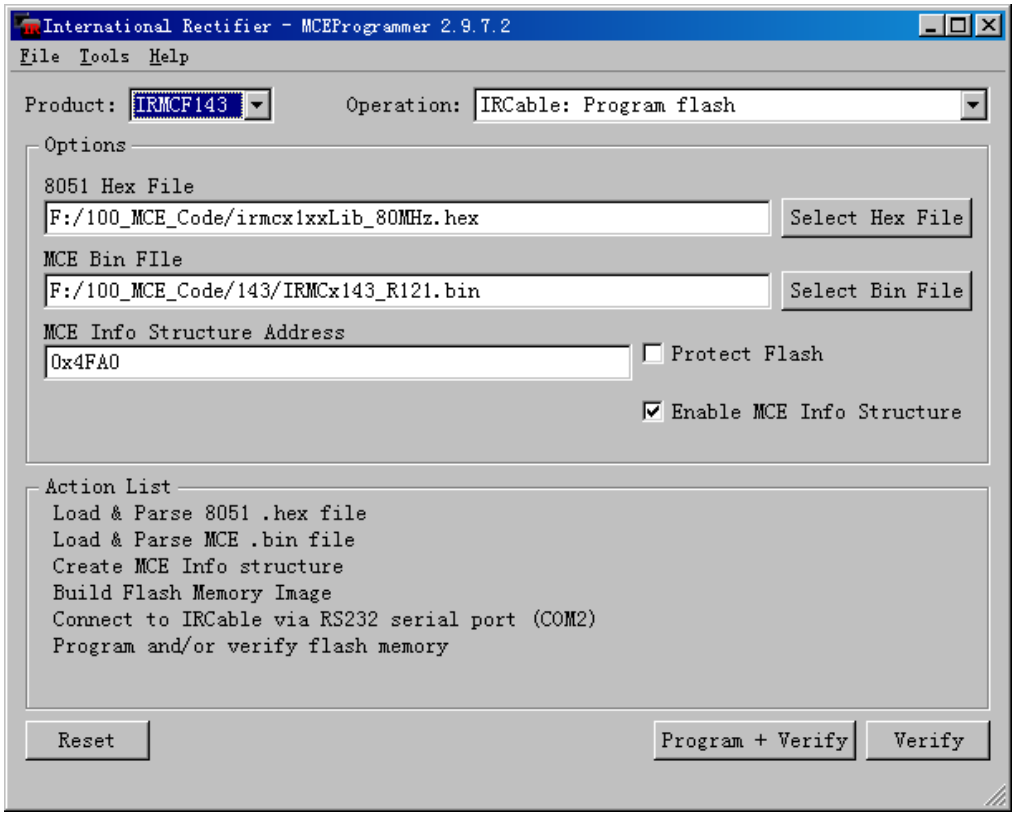


Figure 13. MCEProgrammer Settings for IRMCF1xx

4.3.5 Programming

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

If the “Program + Verify” option is selected, MCEProgrammer reads back the flash 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.

If read back and verify of memory is not required, use the “Program” option. With FS2 JTAG, this option considerably reduces the programming time.

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

4.4 Using the Programming Board

The IRMCx100 Programming Kit (IRMCx100 PROG R1.0 or above version) is the universal IRMCK100 & IRMCF100 programming board, which contains all of the functions of the IRMCK171PROG board and IRMCK143 PROG board, Please refer to the IRMCx100 Programming Kit User Manual for instructions on using IRMCx100 Programming Kit to program IRMCx100 parts.

4.5 Programming On-Board

The IR reference design boards provide the option of on-board programming using IRCable or the FS2 JTAG device. With this option it is not necessary to use a separate board to program OTP memory. This provides a low cost “getting started kit”.

4.5.1 Special Procedure for OTP Programming On-Board

As with the programming boards, on the reference design board VPP must be applied to the chip in order to program OTP memory. The reference design boards have a dedicated 6.75V power supply to handle this task. VPP should not be applied for extended lengths of time and, when applied, it is not possible to use P1.5 as a general purpose I/O pin. Therefore, the boards have a switch that connects VPP to the IC.

Note:
When programming IRMCF1xx flash on the reference design board, *do not* turn on the VPP switch. VPP is required for programming IRMCK1xx OTP memory only.

The procedure to program an IRMCK1xx IC on-board is described below, which shows the IRMCS1043 reference design board as an example. The programming procedure is the same with other reference design boards.

Step 1. Install IC
Install an IC in the socket. Pin 1 must be aligned as indicated in Figure 14.

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} > 120V).

Step 3. Open MCEProgrammer
Open MCEProgrammer and select settings as described in section 4.3.4.

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.

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.

CAUTION: Do not apply VPP until you are actually ready to program. If VPP is applied for an extended period of time it can damage the chip. For the same reason, as soon as you are done programming and verifying, turn off VPP immediately.

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 communication. The programming procedure for this case is the same as described above.

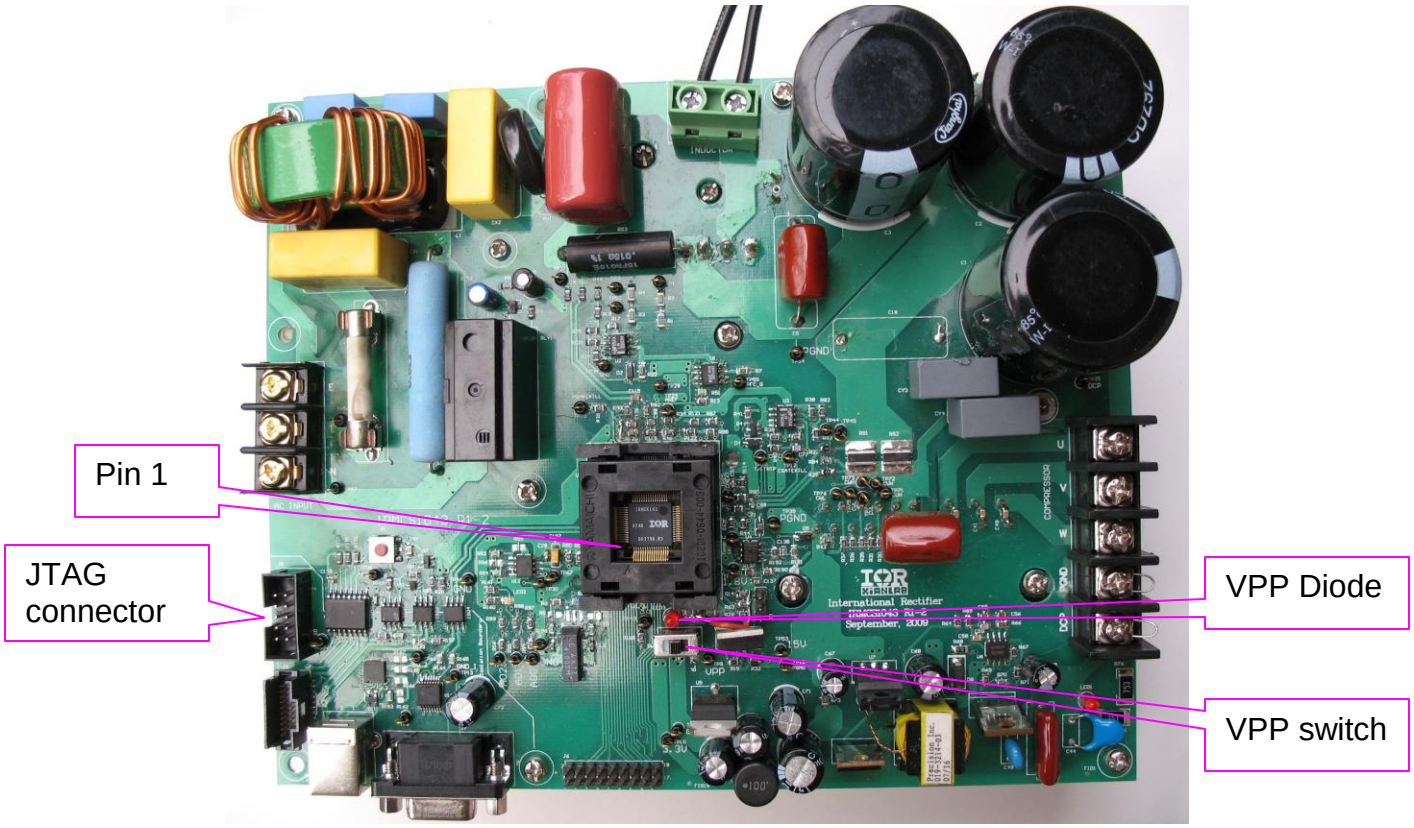


Figure 14. Programming on Board

4.6 Download to RAM

MCEDesigner provides reload of MCE program RAM for all IRMCx100 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 or flash 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.

During the reset/boot process, hardware automatically copies the MCE program from non-volatile memory to MCE program RAM for execution. When using MCEDesigner to reload RAM all the program RAM is overwritten by the data in the specified .bin file. However, since only RAM is loaded (non-volatile memory is not reprogrammed), once the controller has been reset and the boot process executed, any data loaded to RAM will be lost. Therefore the user must reload the MCE program manually after each hardware reset.

To download an MCE program to RAM, go to the “Tools” menu in MCEDesigner. Select “Load Target” and a menu like the one in Figure 15 is displayed.

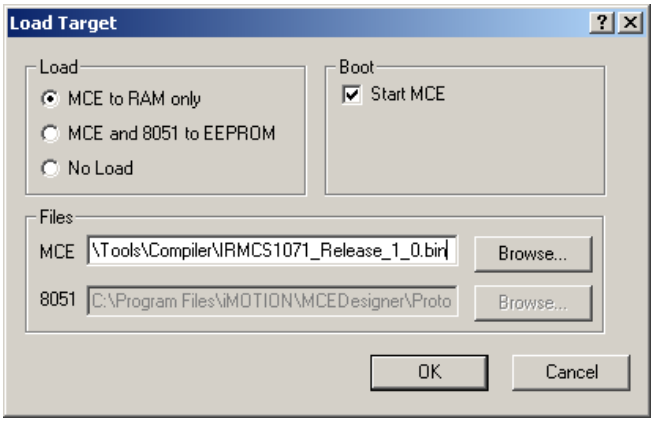


Figure 15. Download to RAM

Select “MCE to RAM only” and specify the .bin file to be loaded. If the “Start MCE” checkbox is checked, the MCE will be automatically restarted when the download process is complete. Normally this option should be checked. Click the “OK” button when ready to start the download.

Appendix 1 Custom Programming Methods

IR provides a programming kit 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 IRMCK100 series OTP memory images. These binary files can then be integrated with any custom OTP programmer. The binary file created contains the full address range of OTP (32K bytes) with address range 0x0000 to 0x7FFF.

JTAG Test Mode

The IRMCK100 series has a single JTAG port that can be used to either debug the embedded 8051 microprocessor 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 IRMCK100 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 dual-purpose pin P1.5/VPP. When 6.75V is supplied to this pin it will act as the supply rail for the internal OTP memory. If performing programming in-circuit it is important to either verify that external components can withstand 6.75V or isolate the P1.5/VPP pin during programming. An easy way to isolate these devices is to have a jumper present on the P1.5 line; after the OTP is programmed the jumper can be put in place.

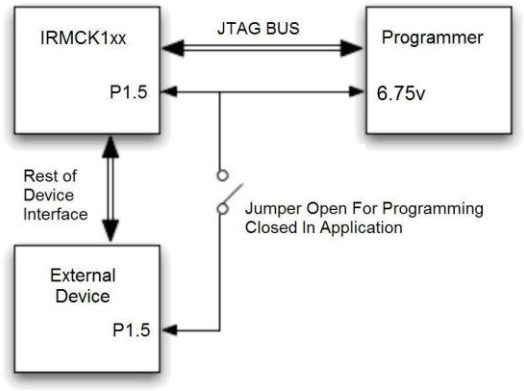
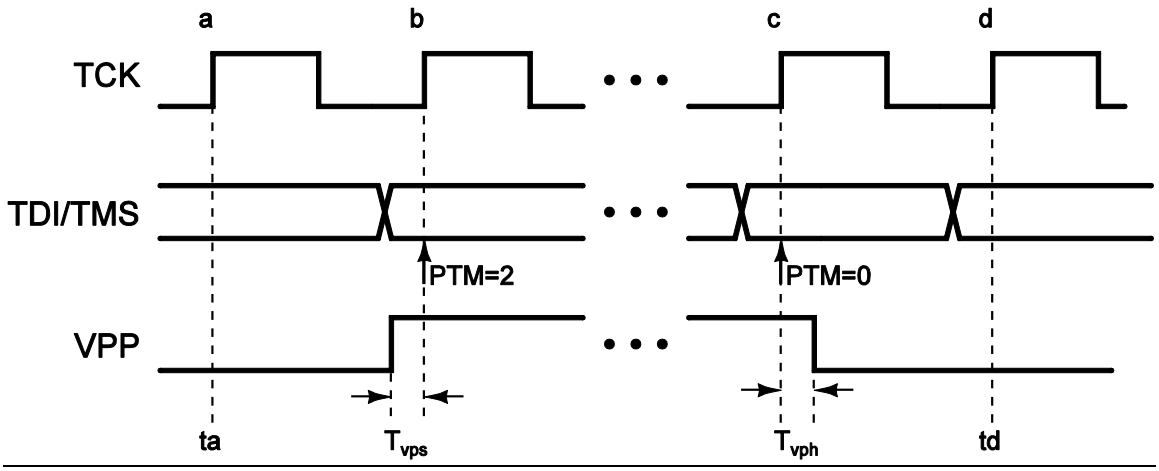


Figure 16. Device Isolation Example

During OTP read mode, the VPP pin can be at either VDD 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 30 seconds can damage OTP memory.



Note: PTM=2 means OTP is configured into program mode; PTM=0 means OTP is configured back to read mode. Tvps = 10ns, Tvph = 10ns.

JTAG Overview

The JTAG interface in the IRMCK100 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 5.

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

Table 5. 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.

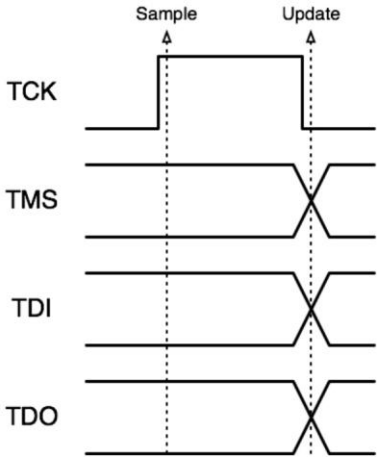


Figure 17. Basic JTAG Cycle

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 6. 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	Description
IR	8 bits	Instruction Register
DR	16 bits	Data Register
OTP_Setup	8 bits	The OTP programming configuration register
OTP_Wr_Timer	8 bits	Number of TCK clocks X 64 to write one byte of data
OTP_JTAG_Address	16 bits	The current OTP address that will be accessed
OTP_Data	8 bits	The data byte read from or written to OTP
Test_Modes	16 bits	Configures the test interface mode

Table 6. 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 7. The OTP_Setup register is written using IR command code 0x50 and read using command code 0x60.

Address Advance			SKIP	POEB	PTM		
7	6	5	4	3	2	1	0

OTP_Setup.7 – OTP_Setup.5	AdAdv	Number of addresses to advance in auto-increment mode (number to advance = 2 ^ AdAdv)
OTP_Setup.4	*Reserved*	Set this bit to 0 for OTP programming and verify
OTP_Setup.3	POEB	OTP Output Enable Bit 0: Enables OTP memory read access 1: Enables OTP memory write access This bit must be set to 1 for OTP programming
OTP_Setup.2 – OTP_Setup.0	PTM	Program Test Mode 000 = Read OTP 010 = Program OTP

Table 7. OTP_Setup Register Definition

OTP_Wr_Timer

This 8-bit 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 0x54 and read using command code 0x63.

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 when IR command code 0x71 or 0x72 is used for the write/read operation. The OTP_JTAG_Address register is written using IR command code 0x51 and read using command code 0x61.

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 0x52 and read using command code 0x62.

Test_Modes
The 16-bit Test_Modes register defines what specific functions are to be performed by the test interface of the IRMCK100 series controller. Only one mode is needed for OTP programming, which is entered by writing 0x0002 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
0x00	Read JTAG TAP controller ID
0xF5	Enter the 'test mode' for OTP memory access
0xF6	Exit the 'test mode' and return to standard 8051 JTAG interface
0x70	Write contents of DR Test_Modes
0x50	Write contents of DR to OTP_Setup
0x51	Write contents of DR to OTP_JTAG_Address
0x52	Write contents of DR to OTP_Data
0x54	Write contents of DR to OTP_Wr_Timer
0x71	Enable OTP Data write (auto-increment mode for OTP programming)

Table 8. IR Write Command List

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

Step 1:
Load IR
Step 2:
Load DR
Step 3:
System will auto-execute instruction with new DR

The 0x71 auto-increment data write command is slightly different in its operation. The auto-increment write mode reduces the number of commands needed during programming. When 0x71 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. Therefore, when programming the IR is set to 0x71 and then DR loads are repeated 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
0x60	Read OTP_Setup to DR
0x61	Read OTP_JTAG_Address to DR
0x62	Read OTP_Data to DR
0x63	Read OTP_Wr_Timer to DR
0x64	Read OTP_Wr_Counter to DR
0x72	Enable OTP data read (auto-increment read mode for OTP verification)

Table 9. 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.

When the 0x72 auto-increment command is set in IR, the OTP_JTAG_Address register is incremented following every DR read. 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.

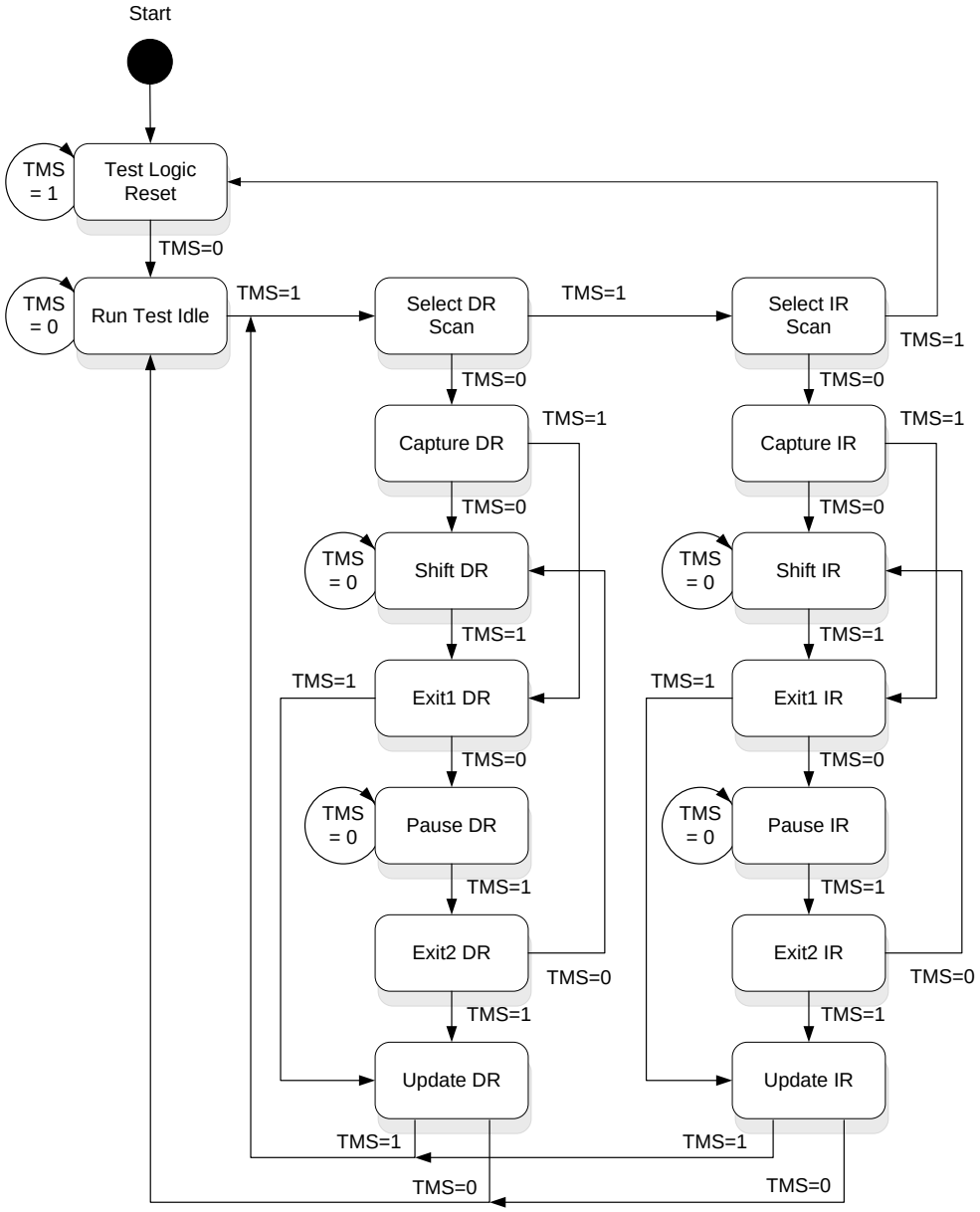


Figure 18. JTAG TMS State Diagram

OTP Programming Example

Write 0xF5 to IR	-	Enter test mode
Write 0x70 to IR then 0x0002 to DR	-	Set TCK to main system clock
Write 0x54 to IR then 0x07 to DR	-	Set OTP_Wr_Timer to 0x07
Write 0x50 to IR then 0x0A to DR	-	Set OTP_Setup to 0x0A
Write 0x51 to IR then 0x0205 to DR	-	Set OTP_JTAG_Address to 0x0205
Write 0x71 to IR	-	Enable auto-increment OTP write
Write 0xA2 to DR	-	Write 0xA2 to OTP address 0x0205
Toggle TCK	-	# of pulses according to OTP_Wr_Timer
Write 0xA3 to DR	-	Write 0xA3 to OTP address 0x0206
Toggle TCK	-	# of pulses according to OTP_Wr_Timer
…		
Write 0xF6 to IR	-	Exit test mode

OTP Verification Example

Write 0xF5 to IR	-	Enter test mode
Write 0x70 to IR then 0x0002 to DR	-	Set TCK to main system clock
Write 0x50 to IR then 0x00 to DR	-	Set OTP_Setup to 0x00
Write 0x51 to IR then 0x7FFF to DR	-	*Set OTP_JTAG_Address to 0x7FFF (last byte)*
Write 0x72 to IR	-	*Italic steps disable OTP read protection*
Read DR	-	*Dummy read*
Read DR	-	*Read last byte of OTP memory*
Write 0x51 to IR then 0x0205 to DR	-	Set OTP_JTAG_Address to 0x0205
Write 0x72 to IR	-	Enable auto-increment OTP read
Read DR	-	Dummy read
Read DR	-	Read OTP address 0x0205
Read DR	-	Read OTP address 0x0206
…		
Write 0xF6 to IR	-	Exit test mode

OTP IR Load Example

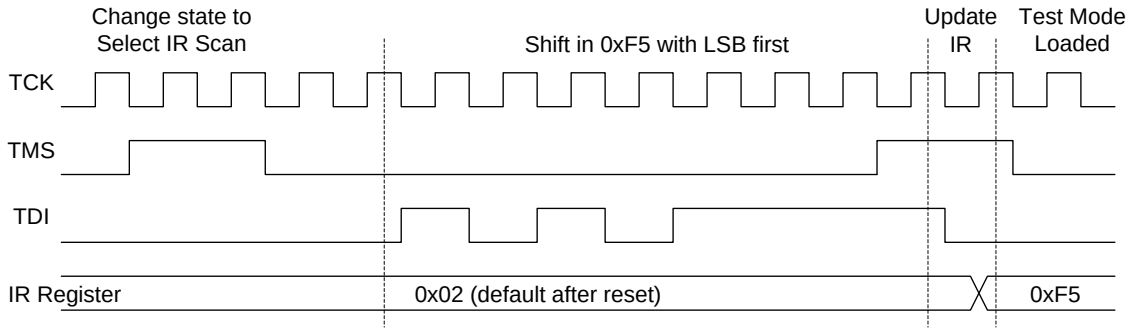


Figure 19. JTAG Load 0xF5 to IR (Enter Test Mode)

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 30 seconds since operation with VPP high for longer periods can damage OTP memory.

Table 10. Critical OTP Programming Timing Information

Appendix 2 MCEDesigner Agent

The MCEDesigner tool is provided with the Reference Design Kit as a development interface to the motor control IC. MCEDesigner consists of a graphical user interface application that runs on a PC and a supporting 8051 application, the "MCEDesigner Agent" that runs on the IC and communicates with MCEDesigner graphical application over the UART interface. The MCEDesigner Agent processes commands from the MCEDesigner graphical application to read and write the IC's motion hardware and motion firmware registers and the registers defined in the Simulink application. The Agent is also responsible for reading and buffering data associated with MCEDesigner's trace monitoring feature.

MCEInfo Structure

The MCE program file (.bin) contains a proprietary header that contains information describing the program. This information includes the name and version of the Simulink design from which the program was created, and the address in MCE program memory to which the program should be loaded.

When MCEProgrammer reads an MCE program file (.bin) and formats it for storage in OTP memory, it strips the header from the .bin file and creates an MceInfo structure, which contains the same information as the header in a slightly different format. The MceInfo structure is stored in OTP memory with the 8051 and MCE programs. The MCEDesigner Agent copies the MCE Info structure to its private data RAM during startup and uses it to initialize the MCE processor and to synchronize operation with the MCEDesigner graphical application.

When MCEDesigner reads an MCE program file and formats it for storage in RAM, it also strips the .bin file header and sends commands to the 8051 Agent instructing the Agent to update its RAM copy of the MCE Info structure to reflect the information associated with the new MCE program being loaded to MCE program RAM.

The MceInfo structure is used by MCEDesigner to verify that the Register Map ID of the .irc file matches the Version ID of the MCE program. If they do not match, then MCEDesigner gives an error. More information on this error can be found in the Application Developer's Guide.

The MceInfo structure is not required by the hardware boot process, but is a standard component of the 8051 MCEDesigner agent software. An 8051 application developed by the user is not required to define or make use of the MceInfo structure, although the structure can be included in the OTP image if the option is selected in MCEProgrammer (see Section 1.1).

The MceInfo structure is defined (in "C" language format) as follows:

```
typedef struct
{
  unsigned char              validation [ IDV_VALID_LENGTH ];
  unsigned char              numIdvDesignIdBytes;
  char                       idvDesignId [ MAX_IDV_DESIGN_ID_LENGTH + 1 ];
  unsigned char              numIdvVersionBytes;
  char                       idvVersion [ MAX_IDV_VERSION_LENGTH + 1 ];
  unsigned char xdata *      pLoadAddr;
  unsigned short             loadSize;
  unsigned char xdata *      pExecAddr;
  unsigned short xdata *     pTraceAddr;
  unsigned char              sysTracePage [ 16 ];
  unsigned short             PageBase [ 8 ];
} MCE_INFO;
```

The relationships between fields of the MceInfo structure and fields of the binary file header are shown in the table below.

MceInfo Field	Binary Header Field	Note
validation	n/a	This field is added by MceDesigner and has no correspondence in the binary file header. MCEDesigner sets this field to the ASCII string "iMOTION" and the 8051 Agent uses it as an indication that a valid MceInfo structure is stored in OTP memory.
numIdvDesignIdBytes	ILEN	The length of the design ID portion of the IDV string (preceding the newline). The length does **not** include the NULL terminator on the idvDesignId string.
idvDesignId	IDV	The design ID portion of the IDV string (preceding the newline character in IDV). The idvDesignId string is NULL terminated.
numIdvVersionBytes	ILEN	The length of the version portion of the IDV string (following the newline). The length does **not** include the NULL terminator on the idvVersion string.
idvVersion	IDV	The version portion of the IDV string (following the newline character in IDV). The idvVersion string is NULL terminated.
pLoadAddr	LOAD	The starting address in MCE Program RAM to which the program is to be loaded, specified as an 8051 memory address.
loadSize	PGMLEN	The size of the MCE program, in bytes.
pExecAddr	EXEC	The starting execution address of the MCE Program, specified as an MCE memory address.
pTraceAddr	TRCBASE	The starting address of trace monitoring data in MCE data RAM, specified as an 8051 memory address.
sysTracePage	RTMAP	The TracePage table (second 16 bytes of RTMAP)
PageBase	RTMAP	The PageBase table (first 16 bytes of RTMAP)

The IDV String

The IDV string has a variable length, specified by the value of the ILEN field. There is no NULL terminator or pad character following the last byte of the string. The RTLEN field of the header immediately follows the last byte of the IDV string. The IDV string identifies the Simulink design that was compiled to create the MCE binary file. It is made up of the name of the model file (minus the ".mdl" extension) and the version number of the model file (which is created automatically by Simulink and updated whenever the model file is saved). Within the IDV string, the design name and version number are separated by a newline character (0x0A). MCEDesigner uses the IDV string to verify that its current database (loaded from a configuration ".irc" file) is consistent with the MCE program loaded to memory on the target platform.

RTMAP

The RTMAP field of the .bin file header contains the PageBase table (8 16-bit words) followed by the TracePage table (16 8-bit words). The PageBase table provides the base addresses of the read and write registers defined in the Simulink design. The registers are divided into eight sections, with a base address for each section. (Depending on the design and the device type,

not all sections may be used.) The optional header file output by the MCE Compiler includes the definition and initialization of a PageBase table containing the same information that is stored in the .bin file header. The TracePage table contains the page number (0 – 3) for each of the sixteen trace data items allowed in the Simulink design. The TracePage table is associated with MCEDesigner's data monitoring feature, and serves no purpose if MCEDesigner is not being used.

Edition 2011-12-23

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email: erratum@infineon.com

Document reference

IRMCx100_SWDevGuide