Application Note AN-1110

Embedded 8051 Application Development for IRMCF/K300 Series Motor Control IC

By Ali Husain, International Rectifier

Table of Contents

Page

INTRODUCTION....2
- Purpose....2
- Requirements....3

GETTING STARTED GUIDE....3
- Software Setup....4
- Hardware Setup....5
- Hardware and Software Startup....6
- Making changes to the sample code....7

SAMPLE CODE STRUCTURE....7
- Clock Frequency....8
- Registers....8
- Files and Functions of Sample Code....10

CONTROLLING A MOTOR....12
- Drive Configuration....12
- MCE Header File....13
- Motor Functions....13

ADVANCED IMPLEMENTATION....14
- Downloading 8051 code to EEPROM....15
- Restoring the original 8051 code....16
- Troubleshooting....16

APPENDIX: Keil uVision2 Project Options....18

This application note describes the 8051 sample code supplied with the IRMCS3041 Reference Design Kit. It includes descriptions of the code to transfer data between the 8051 and MCE cores along with drivers for the communication ports and other 8051 peripheral blocks.

INTRODUCTION

The 8051 microprocessor, included in the IRMCF/K300 series of motion control ICs, 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 IRMCF/K300 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 IRMCF/K300 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.

The 8051 software application controls and monitors the operation of the MCE through a read/write register interface. The 8051 "developer" application used with the MCEDesigner tool does this 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."

Purpose

The purpose of this application note is to describe the implementation of 8051 microprocessor control for use in the IRMCF300 and IRMCK300 series of motion control ICs. This document covers required initializations, settings and functions for 8051 control of the IC. Some examples and sample code are presented. This application note assumes that the user has experience with embedding software programming.

The sample code and 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. After code development and testing with the IRMCF version of the IC, the embedded 8051 code is intended to be written to the ROM of the IRMCK300 series ICs.

Requirements

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

1. FS2 ISA-M8051EW Debugger with Keil uVision driver (FS2 debug pod)
2. Keil PK51 Professional Developers Kit (Keil uVision2)

Using Keil uVision2, the developer can write the control program in the C programming language and then compile it into machine code for download to the IRMCF300 Series IC for testing. uVision2 provides a simulation mode so that portions of the program that are not hardware-dependent can be tested without downloading to the actual IC. The sample source code provided was developed with Keil uVision2.

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. Functions to correctly read and write the shared RAM are included in the sample code. 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.

In addition, the circuit board containing the IRMCF3XX IC should contain the appropriate connectors, drivers, and isolators to interface with the FS2 hardware. If IR's IRMCS3041 Reference Design Board is 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.

GETTING STARTED GUIDE

This section explains how to get started with 8051 application software development using the FS2 debug pod, Keil uVision tools, and the IRMCF300 Series IC. Though this guide applies to a general hardware configuration, often the IRMCS3041 Reference Design Board is referenced as a specific case.

Software Setup

Keil uVision2:

1. Start Keil uVision2. Choose Project→Open Project and open the file IRSamples.Uv2, which is included with the sample code. In the "Project Workspace" window, click on the topmost folder, IRSamples.
2. Choose Project→Options for Target 'IRSamples'. Select the Debug tab.
3. Click the radio button "Use:" and set the field to its right to *Fs2/Keil ISA-M8051EW Driver*, as shown in Figure 1 below. If this option does not appear in the list, the FS2 software has not been installed properly.
4. Click Setting and check that the Settings are: TckRate: *62500* and Tvcc Threshold: *2500*. All the other settings under "Options for Target 'IRSamples' should be configured automatically. The Appendix to this application note lists all the options that should be configured.
5. Press "OK" in the Settings window and then "OK" in Options window.

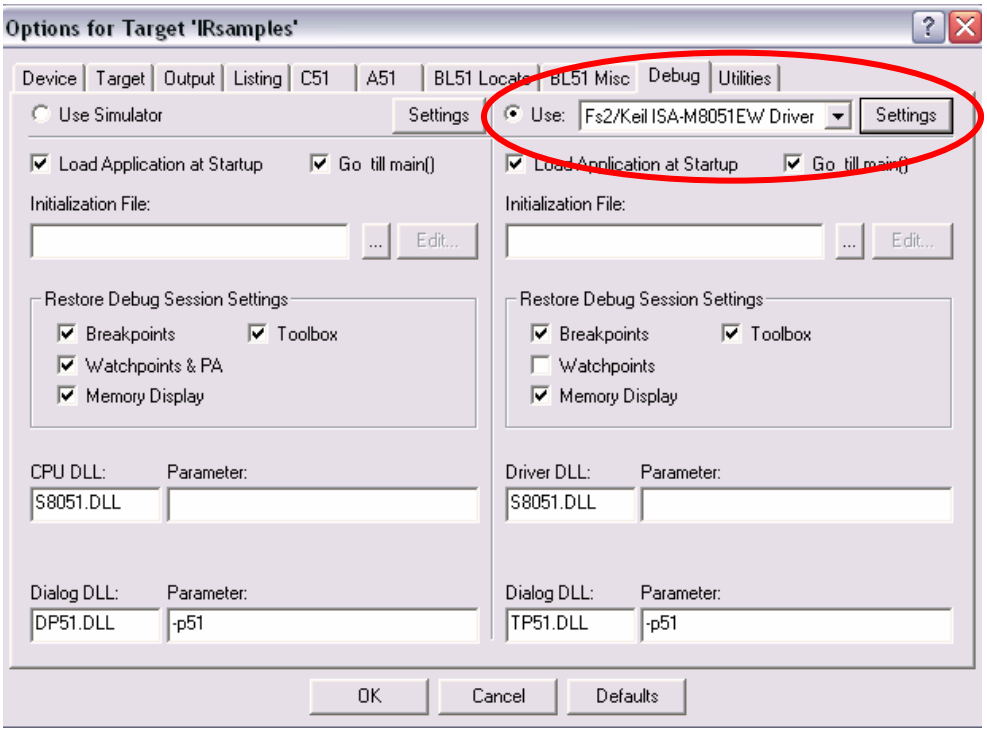


Figure 1: Debug Options window for a uVision2 project

HyperTerminal:

1. Open HyperTerminal (from Windows, choose Start→Programs →Accessories→Communications→HyperTerminal).
2. If the New Connection window does not automatically appear, select File→New Connection. Choose a name and icon for your connection and press "OK."
3. Under "Connect Using," select the appropriate COM port. This will likely be the same port that MCEDesigner uses to communicate with the control board. Press "OK" and the Properties window will open. Set the "Bits per

Second" field to 57600, "Flow control" to None and verify the rest of the settings as shown below in Figure 2.

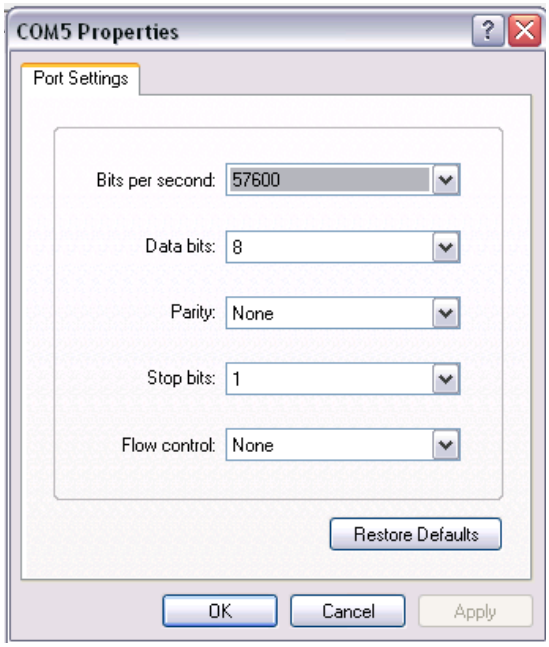


Figure 2: HyperTerminal connection Properties window

Hardware Setup

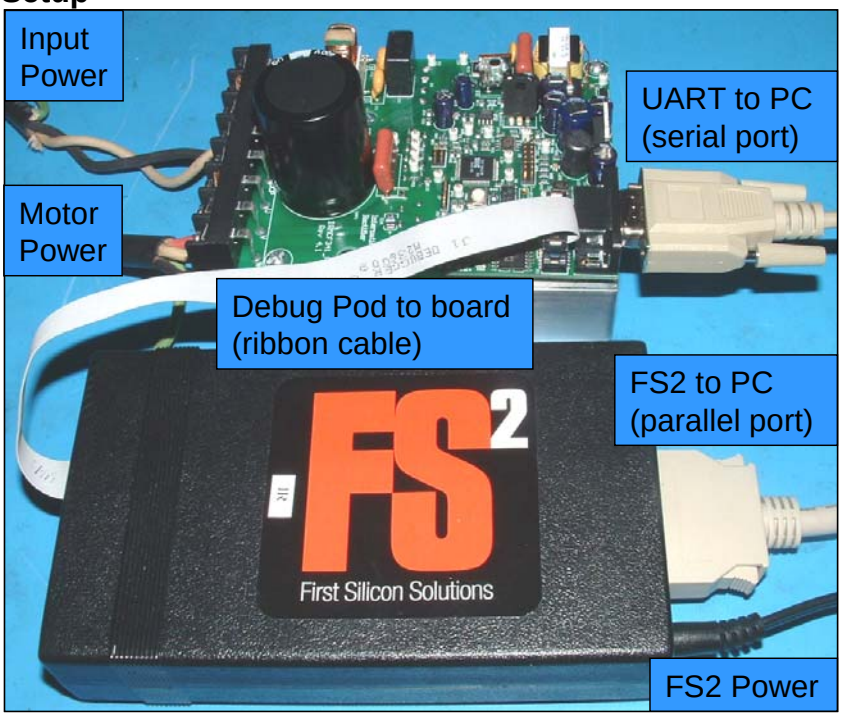


Figure 3: FS2 debug pod connected to IRMCS3041 Reference Design Board

Connect the UART interface to the computer where you have installed Keil uVision2 and MCEDesigner. On the IRMCS3041, this is achieved using an RS-

232 (serial) cable from the control board to the PC. Also, connect the FS2 debug pod to the control board. On the IRMCS3041, the FS2 Pod interfaces to the control IC through the connector J11, as shown in Figure 3 below. If the IRMCS3041 Reference Board is not used, follow 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.

Hardware and Software Start-up

To properly start up the board and software, follow these steps:

1. Apply power to the controller board, and then turn on the FS2 pod.
2. Start Keil uVision2. Choose Project→Open Project and open the file IRSamples.Uv2.
3. 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 4 below.

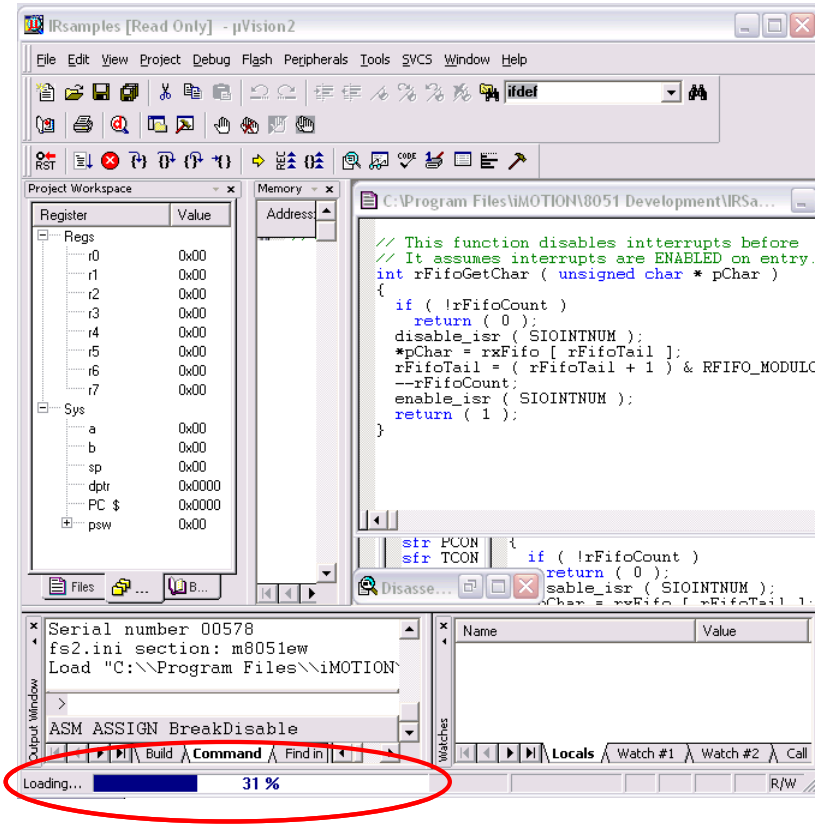


Figure 4: uVision window during 8051 program load.

4. Choose Debug→Go.
5. Start HyperTerminal and open the connection set up above. The motor can be controlled by commands sent through the HyperTerminal. (See section **Motor Functions** for commands.)
6. To exit from Keil after testing, choose Debug→Stop. Then Debug→ Start/Stop Debug Session. The program may be terminated at this time.

Making changes to the sample code

1. To make changes to the sample code, copy all the files of the sample code into a new directory.
2. Select all the files and right-click to select "Properties." De-select "Read-only" and click "OK."
3. Open the new project and modify files as desired. Save files by choosing Files→Save All.
4. To rebuild the machine code, choose Project→Rebuild all target files. The compiler will generate a new .hex file.
5. Follow the instructions in **Hardware and Software Start-up** to test the modified program in hardware. If the hardware is still powered on after testing a previous revision of your program, you can start at Step 3. It isn't necessary to power the hardware off and back on before redownloading.

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 the next section. 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 MCE program counter
3. Configure the drive—write drive parameters to MCE registers for the desired motor
4. Start, stop, 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 drive if necessary.

Clock Frequency

The MCE code must set-up the phase-locked loop to generate the SYSCLK frequency by writing to the SFRs PLLF0 and PLLF1. The sample code already contains instructions to select between 32, 50, 64 and 128 MHz for the clock frequency. Simply uncomment the appropriate #define statement in *timer.h*.

Based on clock frequency the code should set the proper baud rate, timer initialization and reset values. These also are set up correctly when the clock frequency is selected in *timer.h*. However, some drive parameters are also dependent on the clock frequency. The user should regenerate the drive parameters from the "Parameter Configurator" using the new clock rate. See section **Drive Configuration** below.

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 Application Note. A complete list and description of the SFRs can be found in the IRMCx3xx Users Guide. 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 timers
    - Configure the UART
    - Enable analog features such as op-amps
    - Enter and exit low-power modes
    - Configure and enable interrupts
    - Read faults and status
    - Configure and use I^2C/SPI serial interface
    - Read and write the fixed MCE registers
    - Read and write the user-defined MCE registers

    Since SFRs can only be accessed by the 8051microprocessor, writing to these registers is relatively simple. They are defined in *irmcx3xx.h* using a special "sfr" keyword. Each SFR is assigned a name that corresponds to its memory address. In the sample code, the convention is that SFR names are in all capitals. They can be written and read using the name, as with any other variable. Below is an example of the SFR assignments that must be made to set the clock frequency to 50MHz.

```
PLLF0 = 0x62;    // set clock speed to 50 MHz
PLLF1 = 0xC0;
PLLF2 = 1;       // switch to PLL clock
```

```
PLLF3 = 0;
```

2. Fixed MCE Registers (FREGs)—The FREGs are accessible by both the MCE and the 8051 microprocessor. The MCE accesses the FREGs through the Motion Peripheral Blocks, and may modify a subset of them every PWM cycle. The 8051 microprocessor accesses the FREGs through a set of dedicated SFRs. Most FREGs only need to be configured once before running the motor, which can be done by the 8051 application. Additionally, the 8051 application may write to FREGs to start and stop the motor, or adjust parameters due to varying operating conditions.

   Each FREG is defined in *regIf.h* with an alias that begins with "FREG_" and corresponds to its memory offset. The functions **DoRtlRd** and **DoRtlWr** in the sample code are provided to read and write these registers through the dedicated SFRs. It is recommended that these functions be used without modification as the specific sequence of operations is critical for correct operation. Below is an example of writing the number "0" to Fixed MCE Register pwmctrl_1.

   ```
   DoRtlWr ( FREG_pwmctrl_1, 0 );
   ```

3. User-defined MCE Registers (RAM_REGs)—These registers are also accessible by both processors. However, in contrast to the FREGs, the RAM_REGs are not fixed in memory. The RAM_REGs are defined in the MCE Simulink design, and are assigned RAM addresses by the MCE Compiler. The compiler outputs a header file that should be incorporated into the 8051 code (detailed in section **Drive Configuration**). The name assigned to the register has the format:

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

   Although RAM_REGs are directly addressable to the 8051 in shared RAM, another dedicated set of SFRs are used to access them in a controlled fashion that prevents data corruption. (This is necessary because the MCE accesses the registers as a single 16-bit operation while the 8051 requires two 8-bit operations.) The functions **DoRamWr** and **DoRamRd** are provided in the sample code to read and write RAM_REGs through the dedicated SFRs. Below is an example of setting the Motor1 DC Bus Over-voltage Level to 172.

   ```
   DoRamWr ( Motor1_DcBusOvLevel, 172 );
   ```

Note: In MCEDesigner, the FREGs can be distinguished from the RAM_REGs by looking at the "Access" column in the right side of the Motor1 window. To see this column, click on "Register Structure Definitions" on the left side of the Motor1

window. The “Access” column will show Write or Read, optionally followed by a special character, listed below:

* (asterisk)	Fixed MCE Register (FREG)
No character	User-defined MCE Register (RAM_REG)
X	Obsolete Register
+	Local 8051 register. These registers are currently unused.

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 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 16-bit registers in shared RAM with guaranteed coherency using 8051 SFRs. See *RtlRegs.SRC* for the low-level implementation of the FREG interface and *Coherent.SRC* for an implementation of the RAM_REG 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 I2C 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 FREG_FaultFlags register is checked for a fault condition. The interrupt service routine also resets the Watchdog timer. The Watchdog timer must be reset periodically; otherwise the IC as a whole will reset. Timer setup varies based on the clock rate, which is set in *timer.h*.

5. *MceBoot.c* — This file contains functions to initialize the MCE using code that has been programmed to EEPROM by the MCEDesigner tool. It assumes that the automatic boot process has copied the MCE code from EEPROM to shared RAM and an "MCE Info" structure from EEPROM to a fixed location in 8051 program RAM.

   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 preceding the start of the MCE program. 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. For parts that support two UARTs, the code can be compiled for UART1 by commenting out line 20, which defines USE_UART0. 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 SFRs that control the UART.

   flushTx - Initializes the transmit FIFO.

   flushRx - Initializes the receive FIFO.

   setBaudRate - Inializes the baud rate SFR for 57,600 bps, based on the default clock rate of 64 MHz.

   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 the section **Motor Operation**.

   The function **MotorCtrl** checks that the MCE versions defined in regif.c and loaded from EEPROM match, before allowing motor control operations. For more information, see section **MCE Header File**.

8. *RtlRegs.SRC* — Assembly-language functions to read and write RTL registers through the SFR interface.

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.

CONTROLLING A MOTOR

Drive Configuration

After power-up, the motor will not run properly until the MCE has been configured with the correct parameter settings. These drive parameter values are generated using the “Parameter Configurator” (Excel spreadsheet). The second tab of the spreadsheet contains the correctly scaled values for the MCE registers. These values should replace the sample values defined in *MotorCtrl.h*. This process

can be somewhat automated by exporting the sheet in text format and then adding “#define” at the start of each line.

MCE Header File

The MCE code is generated when the Simulink model file is compiled. The compiler also produces a header file (.h) which contains 1) definitions of user-defined MCE registers, 2) register map structures for addressing of the user-defined MCE registers, and 3) product and version identification. This code should replace the samples at the top of *regIf.c* and *regIf.h*. In *regIf.h*, replace the section titled “COMPILER GENERATED DEFINITIONS” with the corresponding section in the MCE Compiler header file. Similarly, in *regIf.c*, replace the section titled “COMPILER GENERATED INITIALIZATIONS” with the corresponding code in the header file.

Note that the MCE design ID and version number of the header file must match that of the MCE code loaded from the EEPROM 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.

The sample code is configured to correctly write to the RAM_REGs of the IRMCS3041 Reference Design Kit and has the proper drive parameters to run the Golden Age GK6040-6AC31 motor.

Motor 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 ones to **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, an X 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. The command character is echoed (sent) back to the host computer on the serial port when configuration is complete. If the motor is running, the command is ignored and an 'X' is sent instead of the command character. See section **Drive Configuration** above.

+

Set forward direction. The command character is echoed when the operation is complete. If the motor is running or in a fault condition, the command is ignored and 'X' is sent instead.

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

F or f
Clear fault condition. The command character is echoed when the operation is complete. If the drive is not in a fault condition, the command is ignored and 'X' 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. The command character is echoed when the operation is complete. If the motor is already running or in a fault condition, the command is ignored and 'X' is sent instead.

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

R or r
Set motor speed. This is a multi-character command. The command character must be followed by exactly four decimal digits (0 - 9) defining the target speed in rotor RPM. If the motor is not running or the requested speed is out of range for the motor (according to the value of define Mtr_Max_Speed) the command is ignored and 'X' is echoed. Otherwise, the operation is performed after all four digits have been received, at which point the command character only is echoed. If a character other than a digit is received, an 'X' is echoed and the command is aborted.

?
Get motor speed. The current speed is read and converted to a single decimal digit (0 - 9), where each digit represents 10% of the total speed range (0 - Mtr_Max_Speed in rotor RPM). The digit is echoed rather than the command character. The get speed command is always valid regardless of the current state.

ADVANCED IMPLEMENTATION

The 8051 code can be used to implement more complex motor control functions than those described above. 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 dependant on the whether the soil setting is

low, medium, or high. Other operations that can be implemented include catch spin and auto-rebalance.

Note that a command interface other than the UART may be used. For example, to receive commands from the digital I/O pins, modify **MotorCtrl** so that it monitors the state of the SFRs corresponding to the appropriate pins and then calls **MotorSeq** as needed.

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

Downloading 8051 code to EEPROM

Once the 8051 application code has been fully tested using the FS2 Pod and Keil uVision, the code may be downloaded to the EEPROM for stand-alone testing. To do so, follow the steps below:

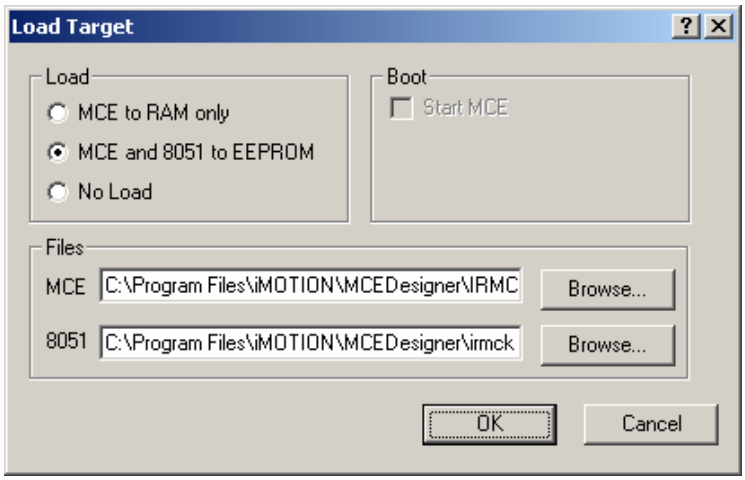


Figure 5: Load Target in MCEDesigner

1. Power down the controller board, then the FS2 debug Pod. Disconnect the FS2 Pod from the controller board.
2. Power up the board and start MCEDesigner. Open an .irc file for your controller board.
3. Click on the System window and then select Tools→Load Target. The Load Target window is shown in Figure 5 above.
4. Select "MCE and 8051 to EEPROM." Choose the appropriate MCE program and, for the 8051 program, the .hex file created by uVision2.
5. When the download is complete (about two minutes), power down the controller board and wait for the COM to go Down.
6. Power up the board, and the COM should stay Down. Close MCEDesigner to release the COM port.
7. Start HyperTerminal (or other UART communication program) and verify that motor control functions operate correctly.

Restoring the original 8051 code

After stand-alone testing, the user may want to restore the original 8051 code to the EEPROM so that application development can be continued with MCEDesigner. To do so, follow the steps below:

1. Power down all equipment. Connect the computer and FS2 Pod to the controller board.
2. Power up the FS2 Pod, then the controller board.
3. In uVision2, open the project corresponding to the MCEDesigner 8051 code (e.g. IRMCx341Lib.Uv2). Choose Debug→Start/Stop Debug Session. Wait for the program to load. Then select Debug→Go.
4. Open MCEDesigner and COM should come Up (green).
5. Click on the System window and then select Tools→Load Target.
6. Select "MCE and 8051 to EEPROM." Choose the appropriate MCE program and, for the 8051 program, the .hex file corresponding to the original code (e.g., IRMCx341Lib.hex).
7. When the download is complete (about two minutes), power down the controller board and wait for the COM to go Inactive. Stop the debugger and end the debug session. Turn off and disconnect the FS2 Pod.
8. Power up the board, and COM should come Up once again. The system is now ready to take commands from MCEDesigner.

Troubleshooting

When a debug Session is started, the message "***error122: AGDI: memory read failed" appears in the Output Window.

- Check that the FS2 debug pod is connected to the parallel port and is turned on.
- Under Project→Options for Target 'IRSamples', click on the Debug tab and then the Settings button and verify that the Comm Port setting is Lpt1 (or the correct one for your PC.)

No characters echoed in HyperTerminal:

- Check that the FS2 debug pod is properly connected to hardware and turned on.
- Under Project→Options for Target 'IRSamples', click on the Debug tab and verify that "Use:" is set to *Fs2/Keil ISA-M8051EW Driver*. Check that the Settings are: TckRate: *62500* and Tvcc Threshold : *2500*.
- Check that the computer running HyperTerminal is connected (by serial cable) to the hardware. Also, verify that HyperTerminal is using the correct port with the correct communication options as described earlier in this document (**Software Setup**). The COM port should be the same one that MCEDesigner uses.

LED does not change from red to blinking green after the drive is configured (c or C in HyperTerminal) (for IRMCS3041)

- Check that DC bus voltage is within range and not causing a fault.

LED changes from red to blinking green (for IRMCS3041) after the drive is configured, but the motor does not turn when commanded.

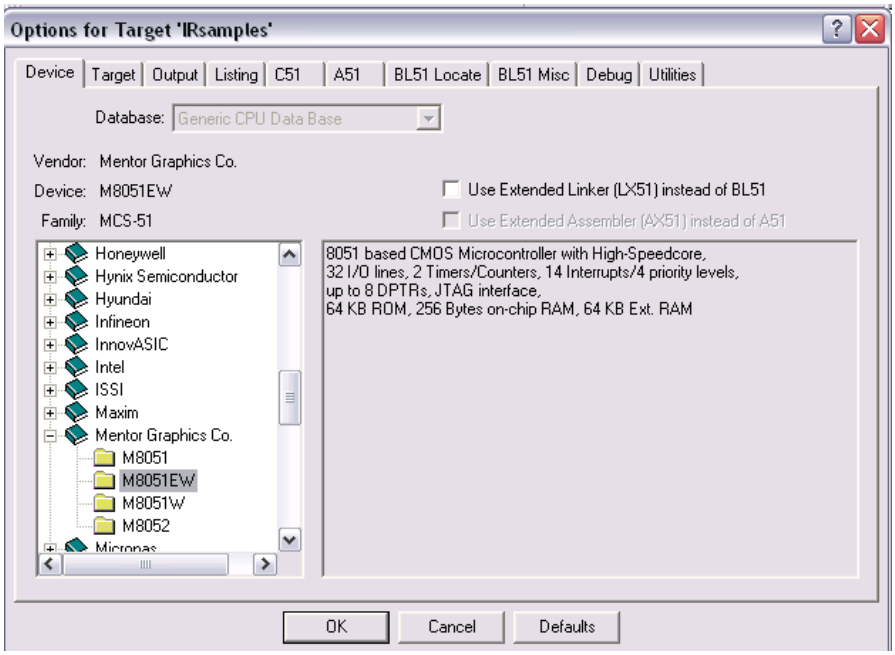
- Check that the correct drive parameters are entered into MotorControl.h
- Check that the MCE is properly started—Set a break point in *MCEBoot.c* to see whether **doMceBoot** is called. If not, the MceInfo structure may be incorrect, or the memory address, RomMceInfo may be incorrect.

APPENDIX

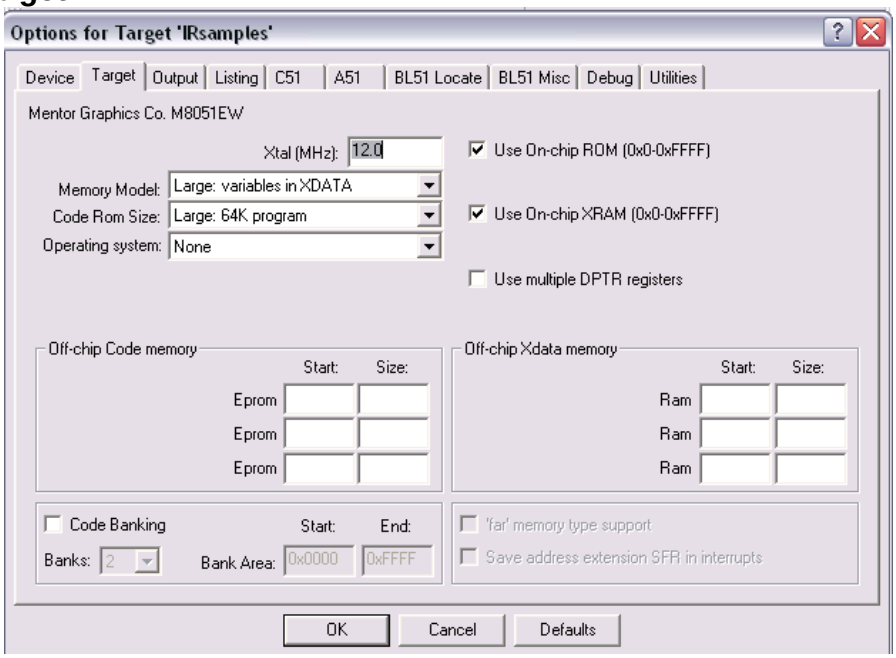
Keil uVision Project Options

These are the options that should be set under Project→Options for Target 'IRSamples'. The uVision2 project file (IRsamples.Uv2) is shipped with all options set as shown below.

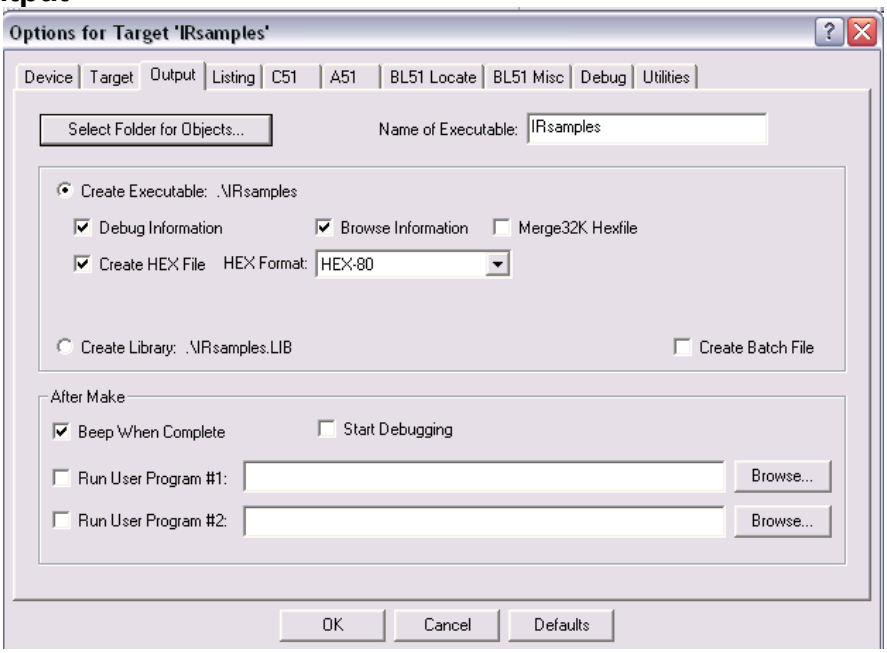
Device



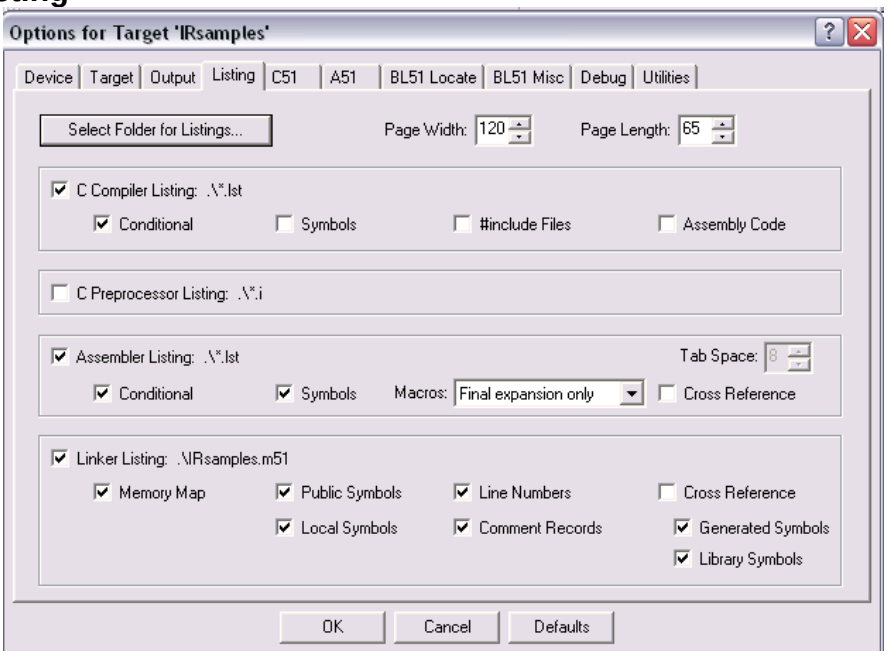
Target



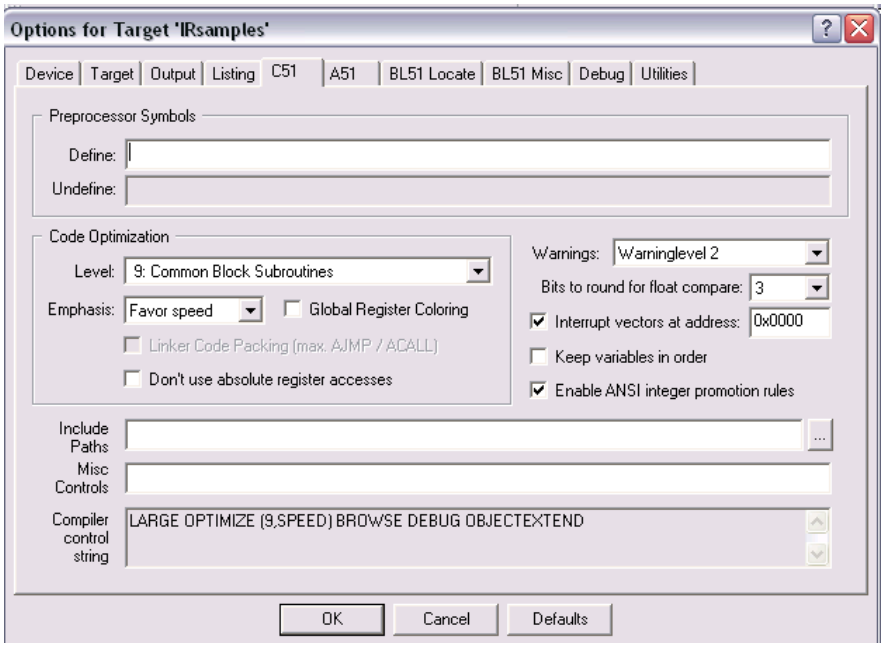
Output



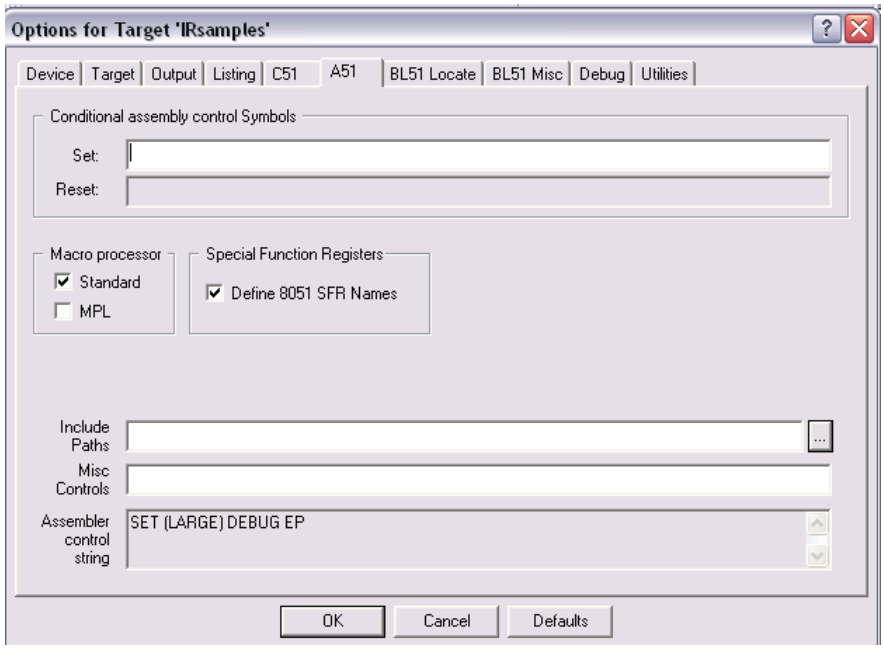
Listing



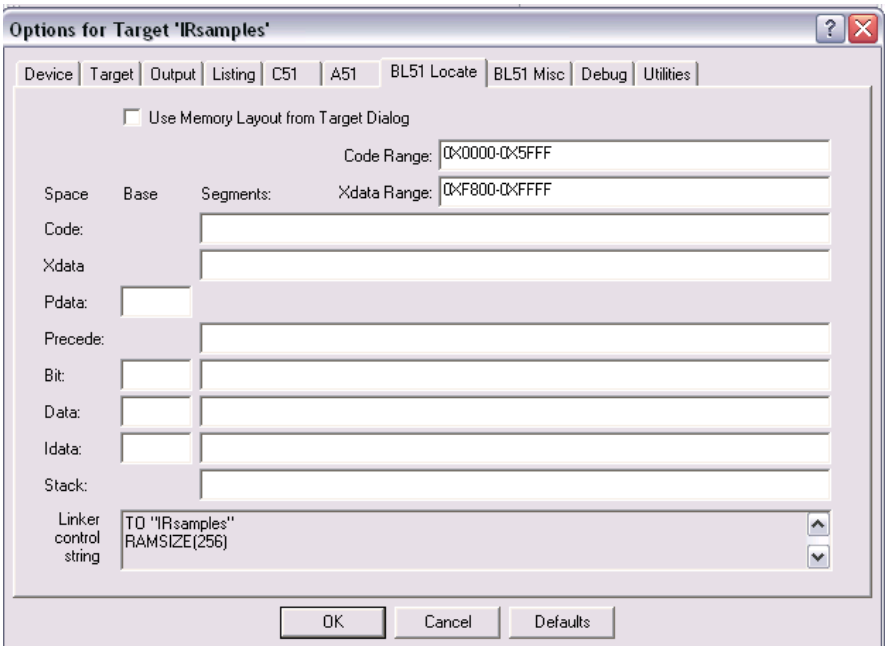
C51


Options for Target 'IRsamples'
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 speed
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,SPEED) BROWSE DEBUG OBJECTEXTEND
OK
Cancel
Defaults

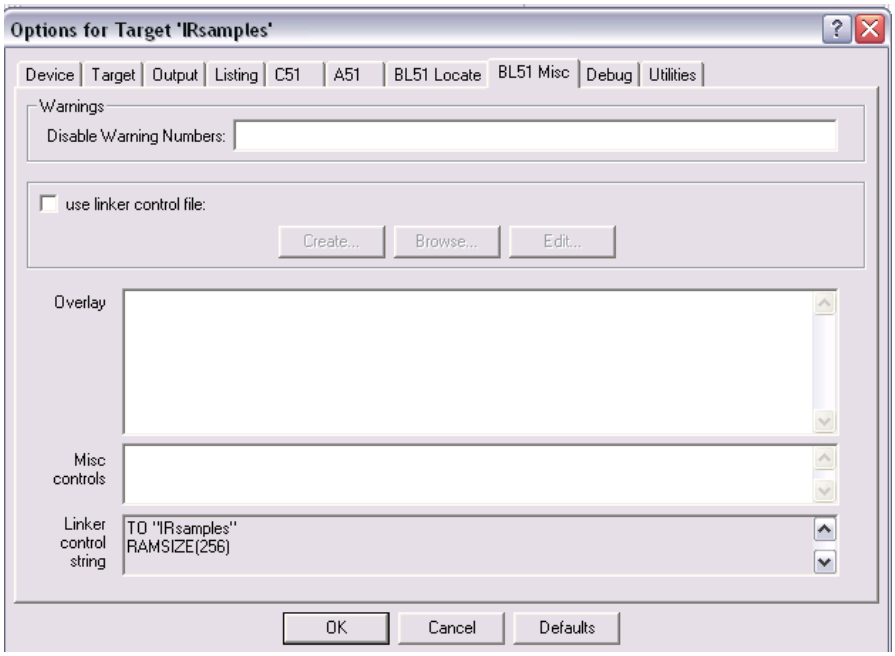
A51


Options for Target 'IRsamples'
Device
Target
Output
Listing
C51
A51
BL51 Locate
BL51 Misc
Debug
Utilities
Conditional assembly control Symbols
Set:
Reset:
Macro processor
Standard
MPL
Special Function Registers
Define 8051 SFR Names
Include Paths
Misc Controls
Assembler control string
SET (LARGE) DEBUG EP
OK
Cancel
Defaults

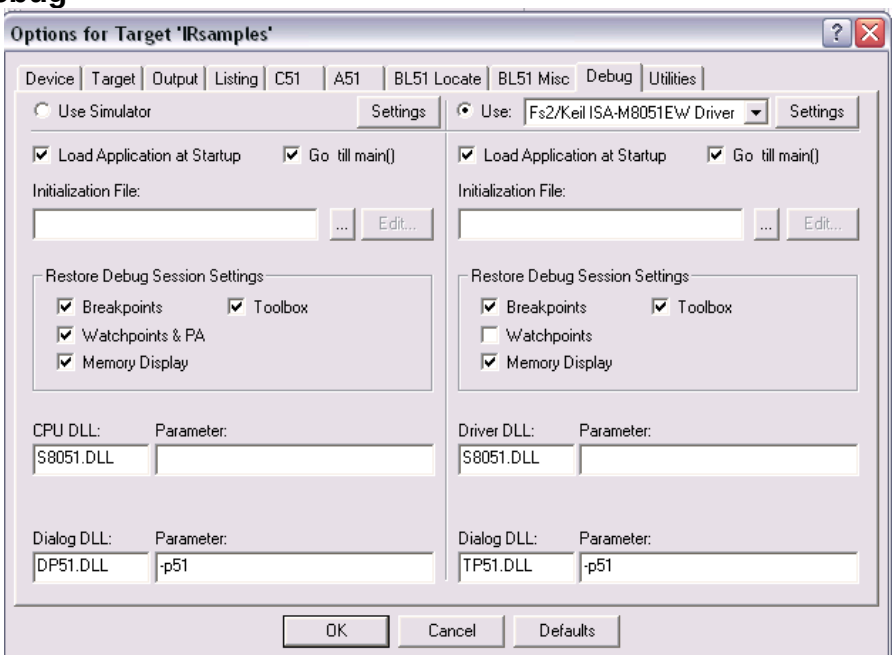
BL51 Locate



BL51 Misc



Debug



Utilities

