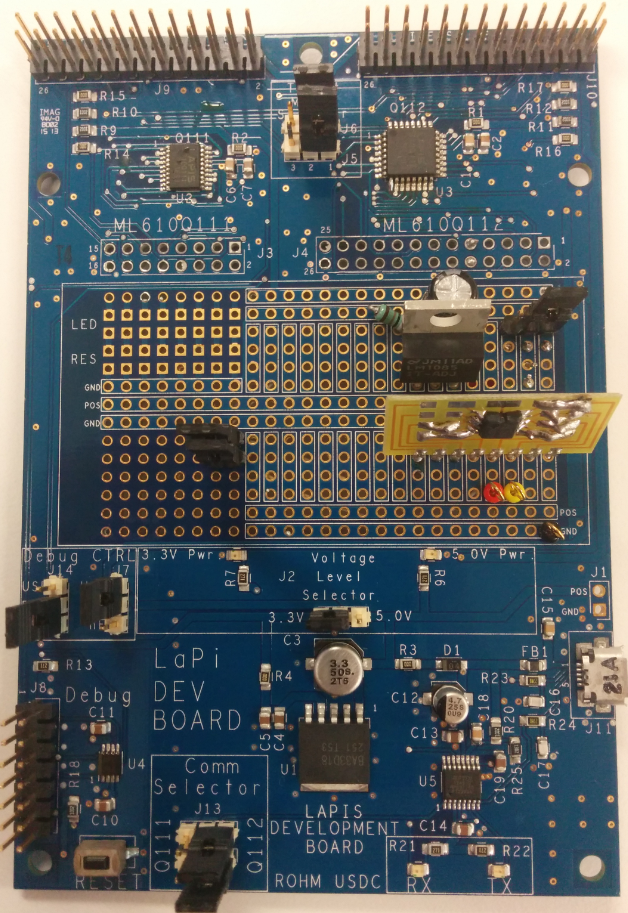
BD99935 I2C Emulator Board Manual



Above: Lapis Development Kit with added EEPROM and LDO for supporting BD99935 pre-testing

15 July 2015, Revision A

Table of Contents

[High Level Block Diagram 3](#_Toc425237368)

[Quick Start Guide 3](#_Toc425237369)

[EEPROM Register Contents 4](#_Toc425237370)

[Hardware Board Explanation 5](#_Toc425237371)

[Jumper Explanations 5](#_Toc425237372)

[Primary Hardware Points 6](#_Toc425237373)

[Firmware Flow Chart 7](#_Toc425237374)

[High Level Flowchart of Operation 7](#_Toc425237375)

[EEPROM Register Content Confirmation 8](#_Toc425237376)

[Method 1: Checking the EEPROM using the on-board I2C Reads 8](#_Toc425237377)

[Method 2: Checking the EEPROM using an Aardvark 10](#_Toc425237378)

# High Level Block Diagram



* The High level block diagram for this application board can be seen above. The left side of this picture shows the blocks used on the existing Lapis Development Kit. The right side of this picture shows the block used on the prototype space of the Lapis Development Kit.

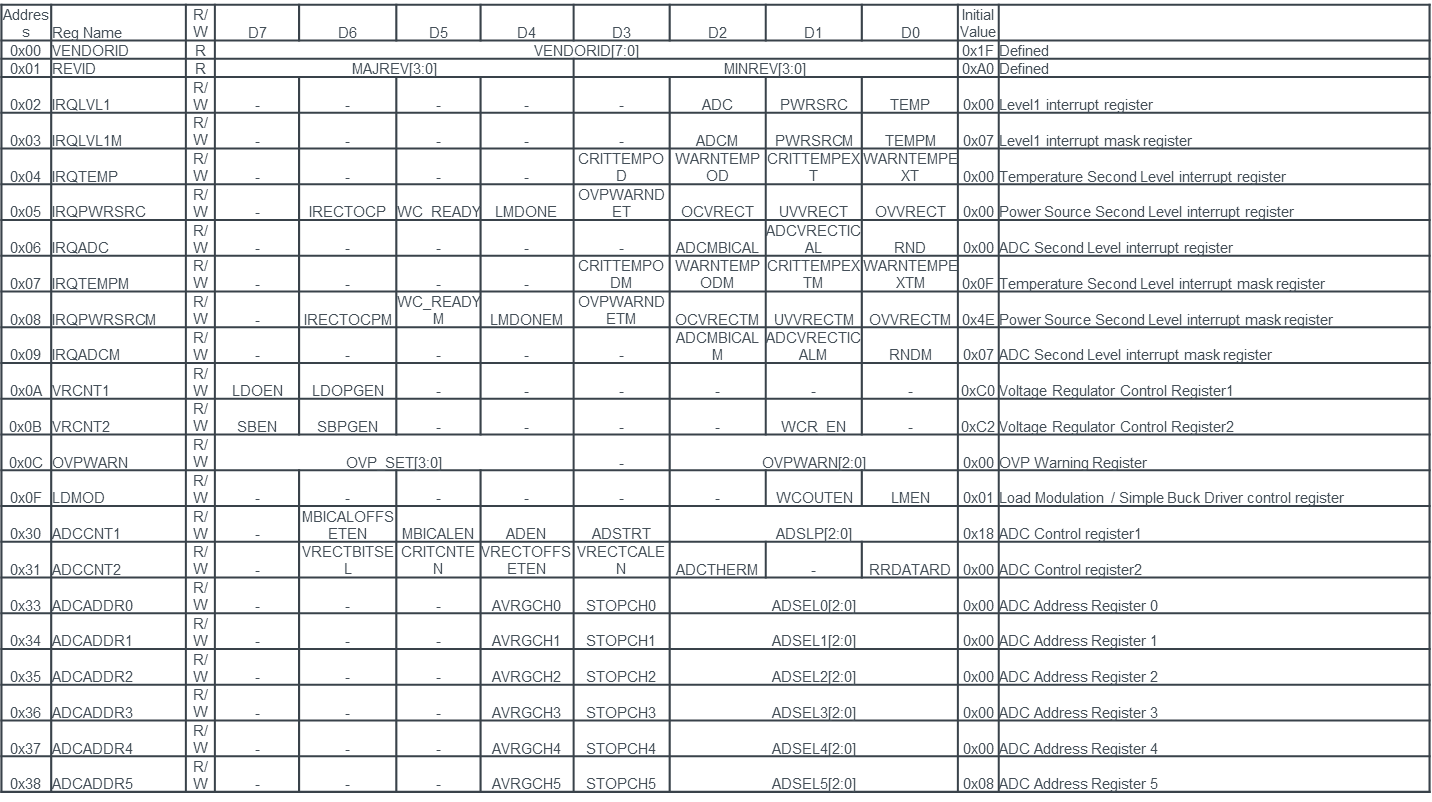
# Quick Start Guide

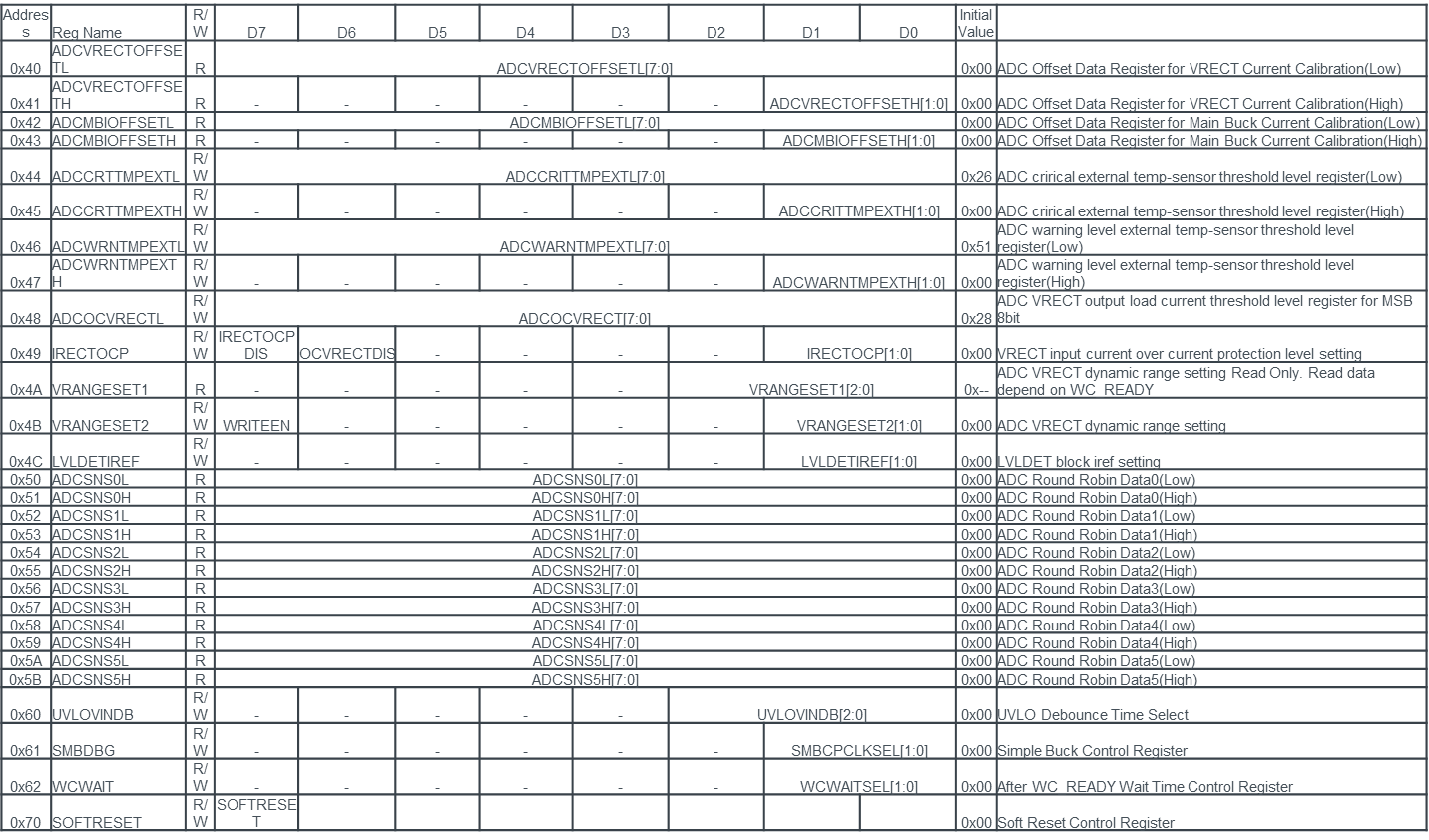
1. The jumpers should be set when we send the board, but jumpers should match the board as per the definitions in the “Hardware Board Explanations” section
2. Power the board using the a uUSB cable to Connector J11
3. Upon power-on the EEPROM will be initialized with the appropriate register map
   * Note: the RESET button can be used to re-initialize the EEPROM register map
4. When ready to connect to the different host, please disconnect the I2C pin jumpers to the Host MCU to ensure the I2C data bus is free.
5. Also, depending on the voltage level, adjust the jumper above the LDO
   * LEFT most Pins = 1.8V
   * RIGHT most Pins = 3.3V
6. Then connect the pins to the board
   * Orange = SDA
   * Yellow = SCL
   * Black = GND

EEPROM Device Address (7bit) = 0x50

IMPORTANT NOTE: This EEPROM requires a TWO BYTE register addressing scheme.

# EEPROM Register Contents





# Hardware Board Explanation

## Jumper Explanations



1. This is a selector Jumper for choosing which MCU to be programmed. For this application this is N/A since the user will not be re-programming the MCU.
   1. J5 = Jump Pins 1 and 2
   2. J6 = Jump Pins 1 and 2
2. This is a selector jumper to choosing how to connect the RESET pin. For this application, we want to use the RESET push button to reset the application if an EEPROM re-write is required.
   1. J14 = Jump USR to RST
3. This is a selector jumper for choosing how to source power to this board. For this application we want to source 5V power from the uUSB connector.
   1. J7 = Jump USR to PWR
4. This is a selector jumper for choosing the voltage level on the “POS” lines of the prototyping space and MCU source power. For this application can use either 3.3V or 5V (but tested under 3.3V condition)
   1. J2 = Jump 3.3V to middle pin
5. This is a selector jumper for choosing the output MCU for the UART lines for either the Q111 or Q112 MCU. This function is not used in this application
   1. Default: Connect Q112 side to middle pin for both jumpers

## Primary Hardware Points



1. uUSB Receptacle
   1. This serves two functions
      1. Power the board using the USB 5V VBUS line
      2. Sets up a UART debug COM port on the host PC
   2. For this application, only 5V Vbus is used
2. Reset Button for the LAPIS Q112 MCU
   1. For this application, this will serve to re-write the EEPROM to the original settings
3. I2C Connection Jumpers
   1. This connects the I2C bus lines to the onboard LAPIS Q112 MCU.
   2. When connecting to an external host, these should not be jumped
   3. When resetting the EEPROM memory map, this should be jumped
4. On-Board Power Selector
   1. This connector allows the user to select the voltage rail of the EEPROM.
      1. Jump Left = 1.8V
      2. Jump Right = 3.3V
5. External Jumpers for I2C Connection
   1. Orange = SDA
   2. Yellow = SCL
6. External Jumper for GND

# Firmware Flow Chart

## High Level Flowchart of Operation

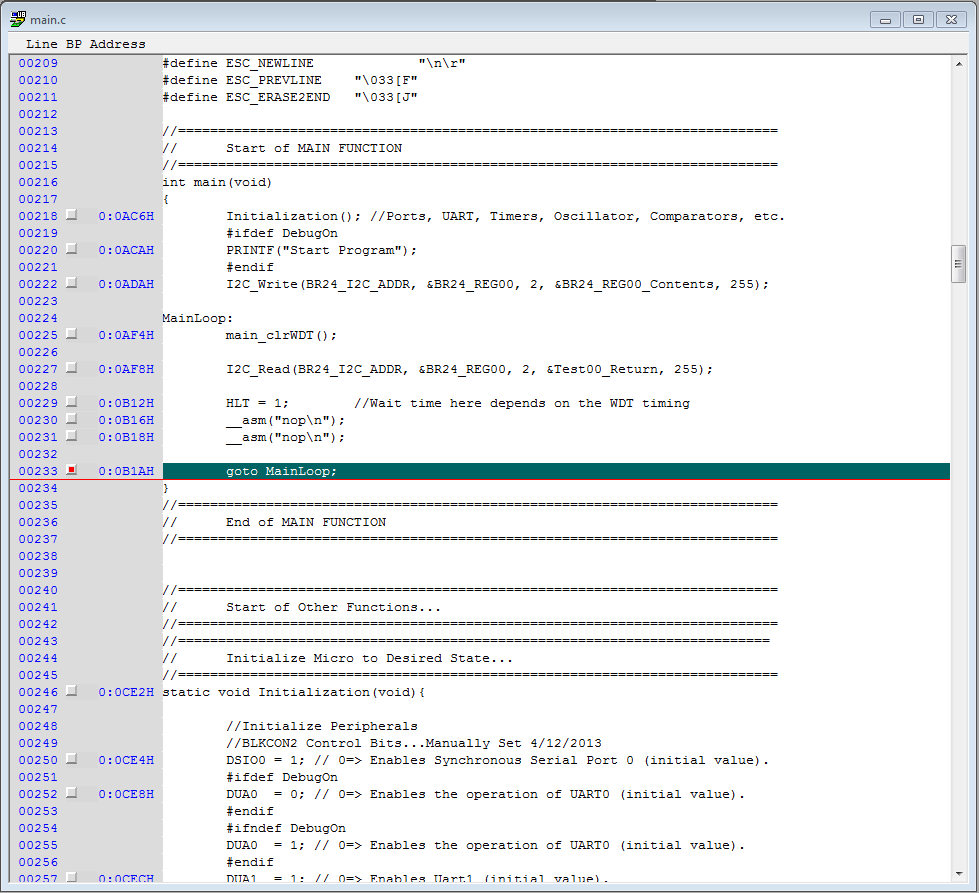


* The Purpose of the MCU on this board is to complete only 1 task: to write the default register mapping to the EEPROM device.
* In the main loop, the EEPROM is read to check the register contents, but ultimately is not helpful to the end user.

## EEPROM Register Content Confirmation

In order to ensure the EEPROM is working as specified, we confirmed operation of this device using two methods.

### Method 1: Checking the EEPROM using the on-board I2C Reads

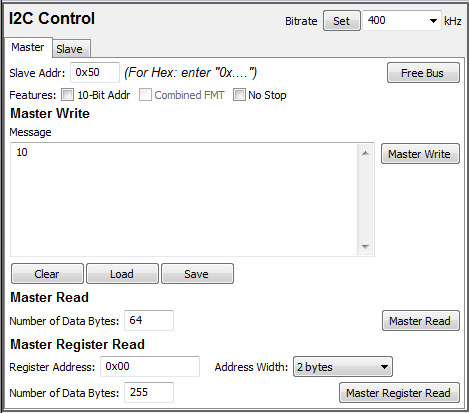


In the above picture, we have setup a breakpoint after the MCU goes into Halt Mode. Thus, at this point, the MCU has written and re-read the register contents of the EEPROM. The Contents are then placed in the character array “Test00\_Return”. Upon checking this variable in the debugger, we can see that the register values were taken successfully



### Method 2: Checking the EEPROM using an Aardvark

In the below picture, the Total Phase Aardvark I2C tool was used to confirm the operation of this device



After Performing the full register map read, we can see that the contents are loaded successfully.

