A31L12X

Current Sensing Board Design Guide using Low Power MCU

Application Note

Version 1.02

Contents

List of Figures

1 Introduction

A31L12X Current Sensing Board Design Guide introduces a method to design a current sensing board using an A31L12X that is a 32-bit Ultra-Low Power MCU. This document intends that users should design the current sensing board to measure the operating current of an application board having A31L12X and check the operating current of the A31L12X.

The Ultra-Low Power MCU A31L12X reduces the operating current by utilizing DEEP-SLEEP mode. During DEEP-SLEEP mode, it consumes about 1uA with 3V power source. For detailed information of A31L12X specification, please refer to **23.11 Supply Current Characteristics** in A31L12X User's Manual.

To measure the current in uA unit on the Current Sensing Board, a high precision 24-bit ADC and a high precision Current Sense Amplifier are used. This document describes how to use them.

In addition, more materials and corresponding solutions of hardware and software that are required for the Current Sensing Board design are provided with this document.

2 Current Sensing Board Block Diagram

Figure 1 shows a functional block diagram of the Current Sensing Board.

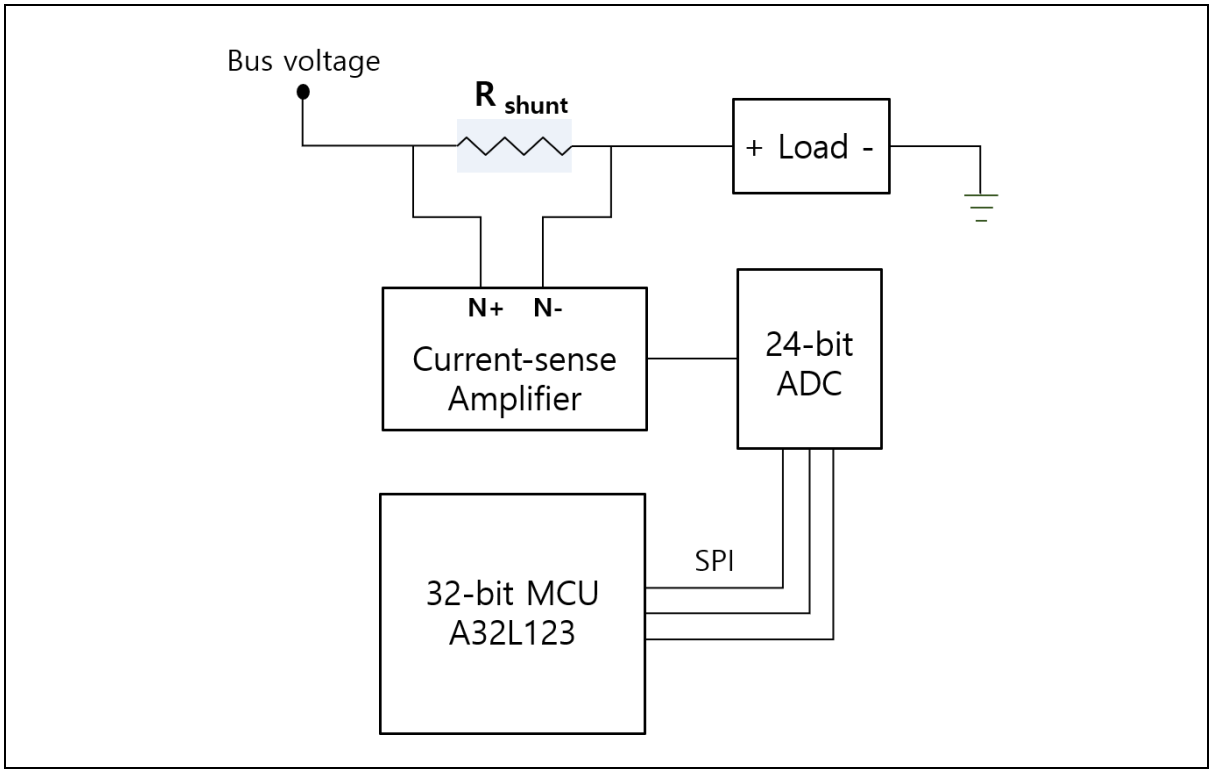


Figure 1. Functional Block Diagram

3 Hardware Description: Parts and Circuits

In Chapter 3, parts and circuits of the Current Sensing Board are introduced.

3.1 Current-Sense Amplifier (INA190)

The Current-Sense Amplifier (INA190), also called a current-shunt monitor, is commonly used for the precision current measurement. It can correctly measure the current using a small Shunt (sensing) resistor, while consuming low power.

For detailed information of this part, please refer to INA190 Technical Document.

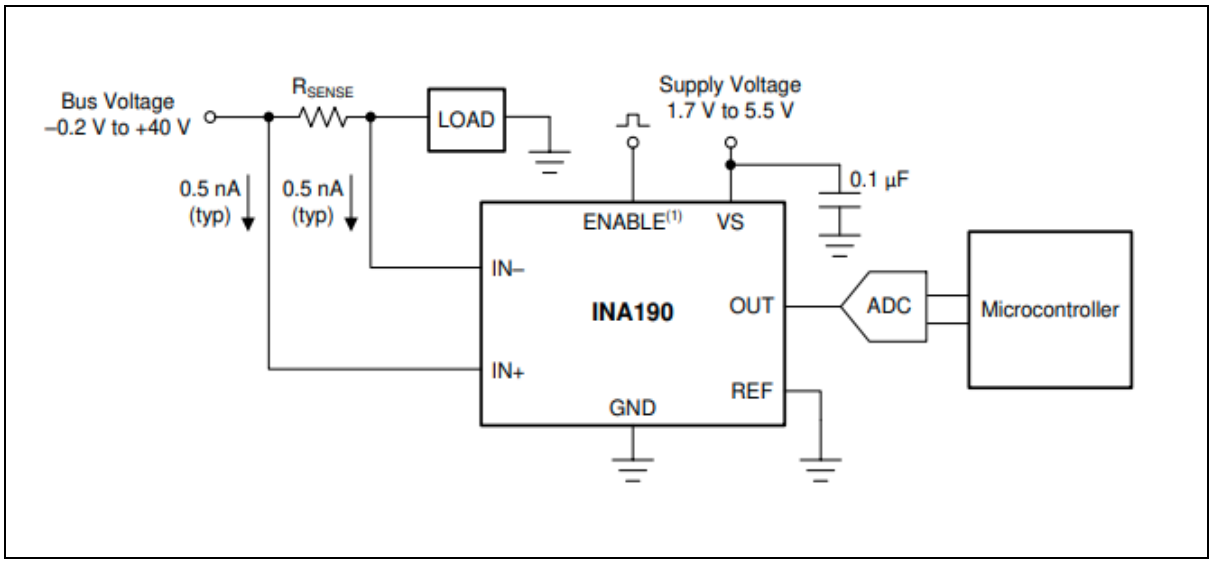


Figure 2. Current Sense Amplifier (INA190) Block Diagram

Figure 2 shows a basic circuit configuration of the Current-Sense Amplifier. Users who use this part for circuit designs need to consider the followings:

1. It can be applied only for one-way of the current that is measured in the circuit. Please refer to the INA190 Technical Document for the two-way application.
2. The most important thing is that a Current Sense Resistor and the INA190 must be closely arranged, and input pin (IN+ and IN-) and the Current Sense Resistor are connected using Kelvin Connection.

3. Users need to determine the range of the current to sense before selecting the resistor value of R_{sense} and the INA190 GAIN.

① Using two formulas, example values of the R_{sense} and INA190 Gain are determined:

$$R_{sense} < \frac{PD_{MAX}}{{I_{MAX}}^2} \qquad R_{sense} < \frac{V_{DD}}{GAIN \times I_{MAX}}$$

- PD_{MAX} is the maximum allowable power dissipation in R_{sense}.
- V_{DD} is a supply voltage, output limitations that is the gained up voltage difference from the IN- to the IN- pins.
- $GAIN$ is the gain of the Current-Sense Amplifier.

② If users determine the range of the current to measure from 0uA to 100uA and set the values as follows,

- I_{MAX} = 100uA, PD_{MAX} = 0.5W, V_{DD} = 3.3V, GAIN = 500

the resistor value of R_{sense} is calculated as R_{sense} < 66Ω.

As a result, the Current Sensing Board for uA measurement uses a resistor of 50Ω.

③ If users determine the range of the current to measure from 0mA to 100mA and set the values as follows,

- I_{MAX} = 100mA, PD_{MAX} = 0.5W, V_{DD} = 3.3V, GAIN = 50

the resistor value of R_{sense} is calculated as R_{sense} < 0.66Ω.

As a result, the Current Sensing Board for mA measurement uses a resistor of 0.5Ω.

④ For the example of ②, output ranges from 0V to 2.5V and has 25mV resolution per 1uA. For the example of ③, output ranges from 0V to 2.5V and has 25mV resolution per 1mA. Therefore, the Current Sensing Board needs an ADC having 16-bit or more accuracy for the precision current sensing.

⑤ As explained through ① to ④, values of R_{sense} and INA190 GAIN are determined according to the current range that is to be measured.

4. The one-way application of the INA190 described in step 1 has the following layout guideline:

   ① As shown in Figure 3, a bypass capacitor of the power source must be placed close to the power and ground pin of the INA190. The value of the bypass capacitor is recommended as 0.1uF.

   ② The impedance generated in the wiring is minimized when the connection length between IN+/ IN- and the Current Sense Resistor is short.

   ③ To reduce the current measurement error, the Current Sense Resistor and the Input (IN+/ IN-) are connected using Kelvin Connection or 4-wire Connection.

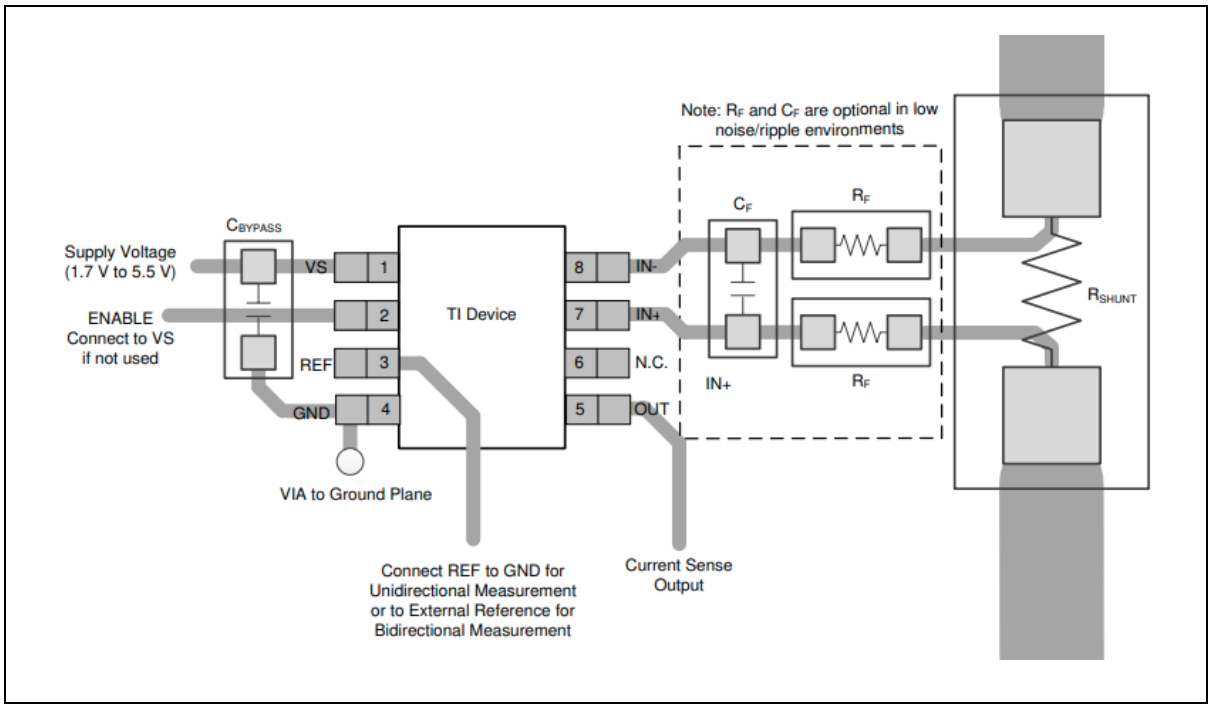


Figure 3. Current Sense Amplifier Circuit Layout Example

3.2 24-Bit Analog to Digital Converter (LTC2400)

The 24-bit Analog to Digital Converter is used to convert the precision analog voltage signal to digital signal. Small voltage signal across the small Shunt (Sensing) resistor is sensed to measure the exact micro current.

For detailed information of this part, please refer to LTC2400 Technical Document.

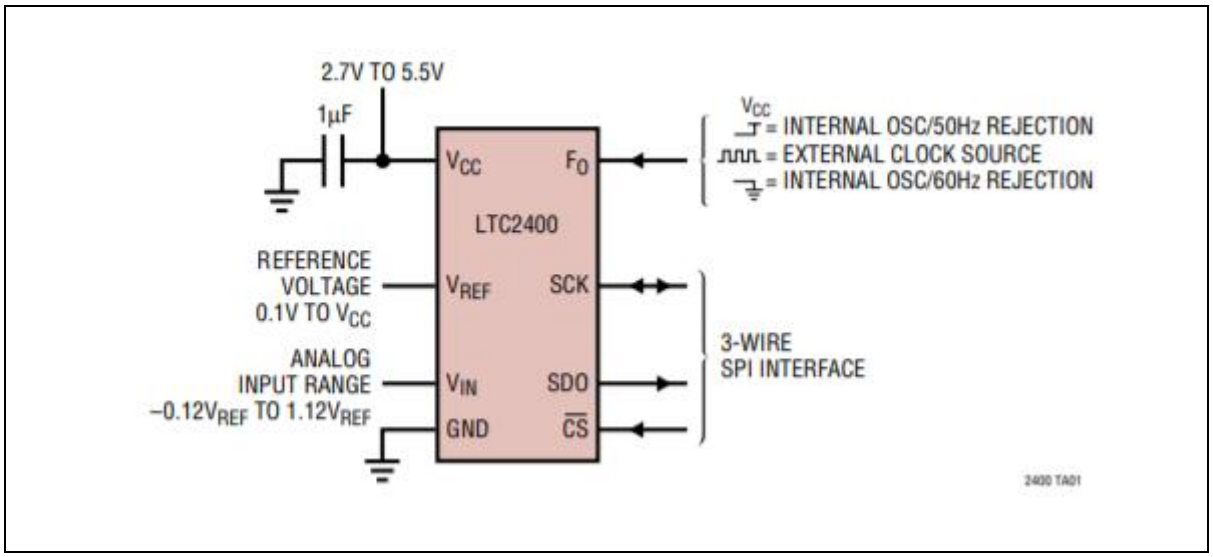


Figure 4. 24-bit ADC (LTC2400) Block Diagram

Figure 4 shows a basic circuit configuration of the 24-bit ADC. Users who use this part for circuit designs and software (SPI communication) configuration need to consider the followings:

1. A bypass capacitor of 1uF should be placed close to the power supply unit.
2. The reference voltage range of the ADC can be determined by using V_{REF}. If users want the input range of the ADC from 0V to 3.3V, users need to apply the V_{REF} = 3.3V.
3. F_O in Figure 4 can be used with 3 alternative functions. In the example circuit in Figure 4, F_O is connected to GND to use an internal OSC of the LTC2400 and a notch filter of 60Hz.

4. The LCD2400 SPI control has the following guidelines:

① If using the internal OSC option, the SCK is used as a digital output pin, and external pull-up resister of 10kΩ is required (in the example circuit, MCU internal pull-up mode is used).

② For the case of using the internal OSC option, SPI communication operation of the internal SCK is described in Figure 5.

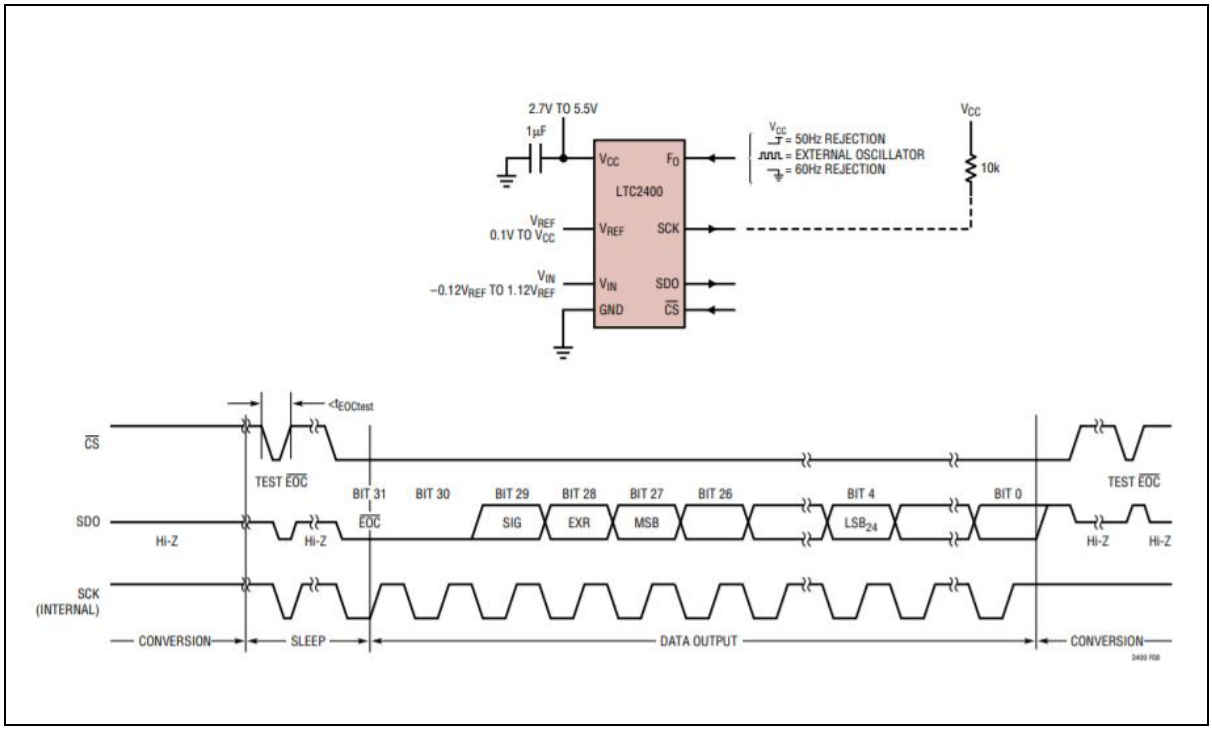


Figure 5. Internal SCK, SPI Comm. Operation 24-bit ADC (LTC2400)

③ When using the internal SCK, the SCK of the LTC2400 is digital output, and the MCU (A31L12X) becomes Slave to receive the SCK as its Input.

④ When using the internal SCK, the SDO of the LTC2400 is digital output, and the MCU (A31L12X) becomes Slave to receive the ADC data through the MOSI pin.

⑤ On the other hand, the CS of the LTC2400 is input and the MCU (A31L12X) becomes Master. Its output signal controls specific timing and allows to start communication through the SCK and SDO of the LTC2400.

⑥ In Figure 5, if CS turns to LOW, data that ADC completes to convert is transmitted through the SCK and SDO.

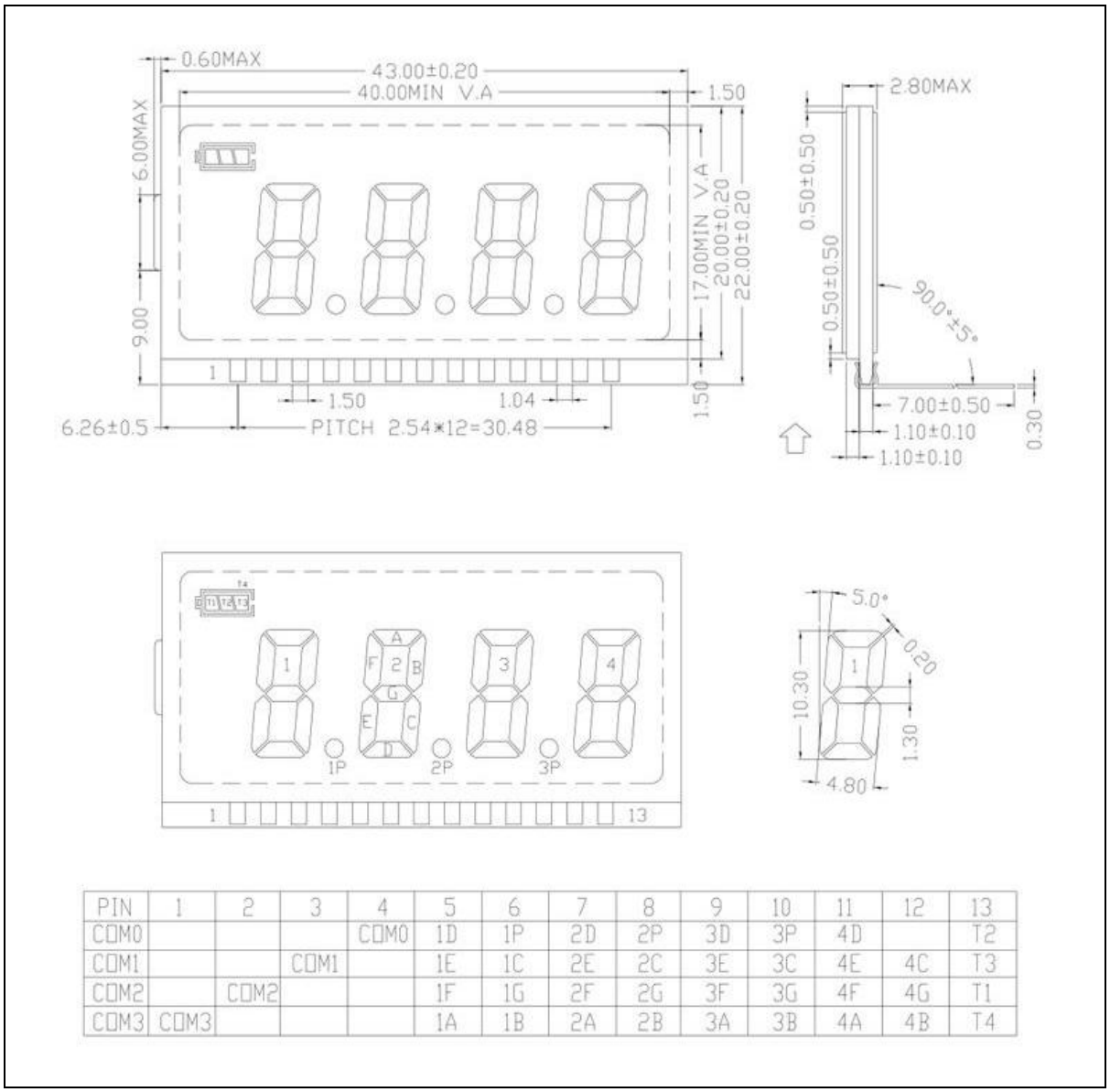
⑦ In Figure 5, BIT 28 to Bit 31 of the SDO data format indicate ADC status, and BIT 4 to BIT 27 are 24-bit ADC's converted value. BIT 27 is the MSB and BIT 4 is the LSB.

3.3 LCD Module and Interfaces

The LCD module and interfaces are used for confirming the control operation and checking output values.

The LCD module uses 4-digit 7-segment LCD to display the measured current value in uA or mA units including decimal points. The interfaces such as button that can start or stop the current measurement is provided for the users to control the current measurement. For detailed information of the LCD part, please refer to GDC8310 Technical Document.

The LCD module and interfaces are free to change for many different applications. Examples are introduced in this section.



PIN	1	2	3	4	5	6	7	8	9	10	11	12	13
COM0				COM0	1D	1P	2D	2P	3D	3P	4D		T2
COM1			COM1		1E	1C	2E	2C	3E	3C	4E	4C	T3
COM2		COM2			1F	1G	2F	2G	3F	3G	4F	4G	T1
COM3	COM3				1A	1B	2A	2B	3A	3B	4A	4B	T4

Figure 6. 4-Digit LCD Panel (GDC8310)

Figure 6 shows a basic circuit configuration of the LCD module. Users who use this part for LCD control need to consider the followings:

1. The LCD module in Figure 6 can be controlled by using LCD driver peripherals of A31L12X.
2. The GDC8310 operates with 3.3V of operation voltage and uses 1/4 Duty 1/3 Bias drive techniques.
3. Each of COM and SEG of A31L12X and GDC8310 can be connected by matching them as shown in Figure 7.

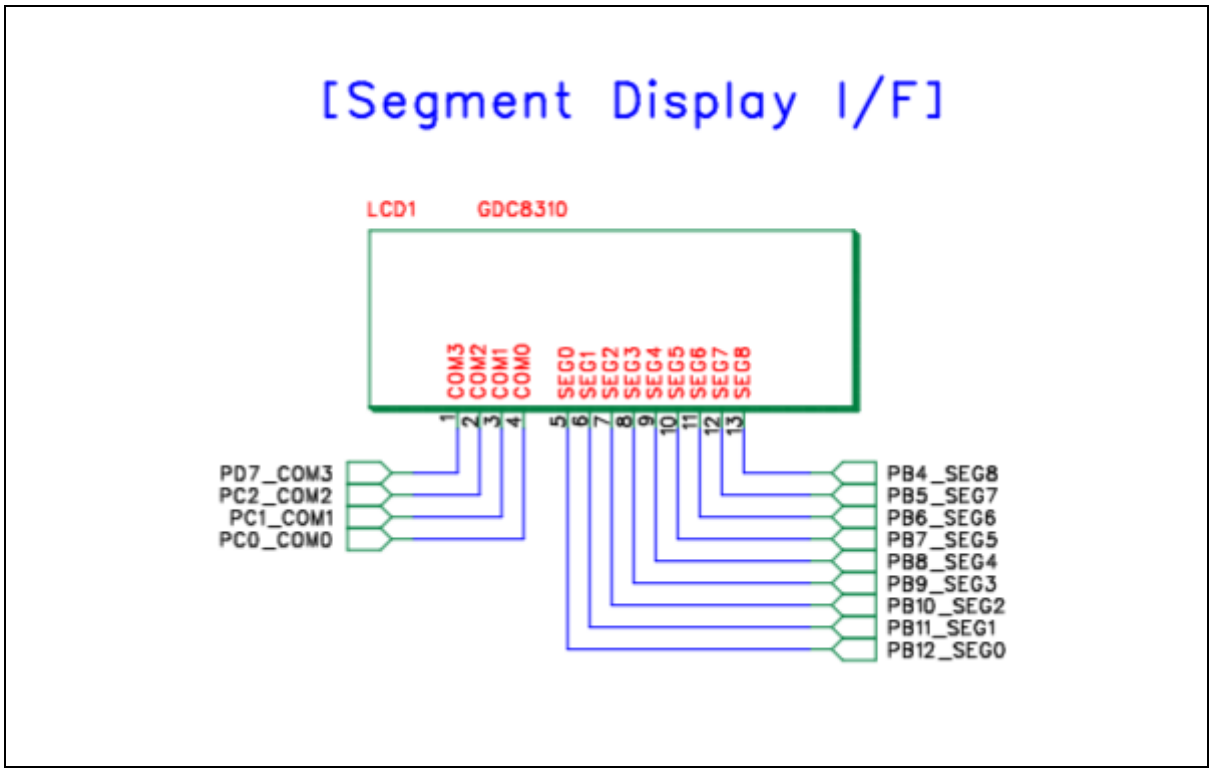


Figure 7. Connections Circuit Diagram of A31L12X and 4-Digit LCD Panel

Finally, a button is added to start or stop the current sensing measurement. For detailed information of the control method, please refer to example code in Chapter 4.

4 Software Structure

In Chapter 4, knowledge that software developers must have is described. Figure 8 shows a State Machine diagram of the software that drives the Current Sensing Board.

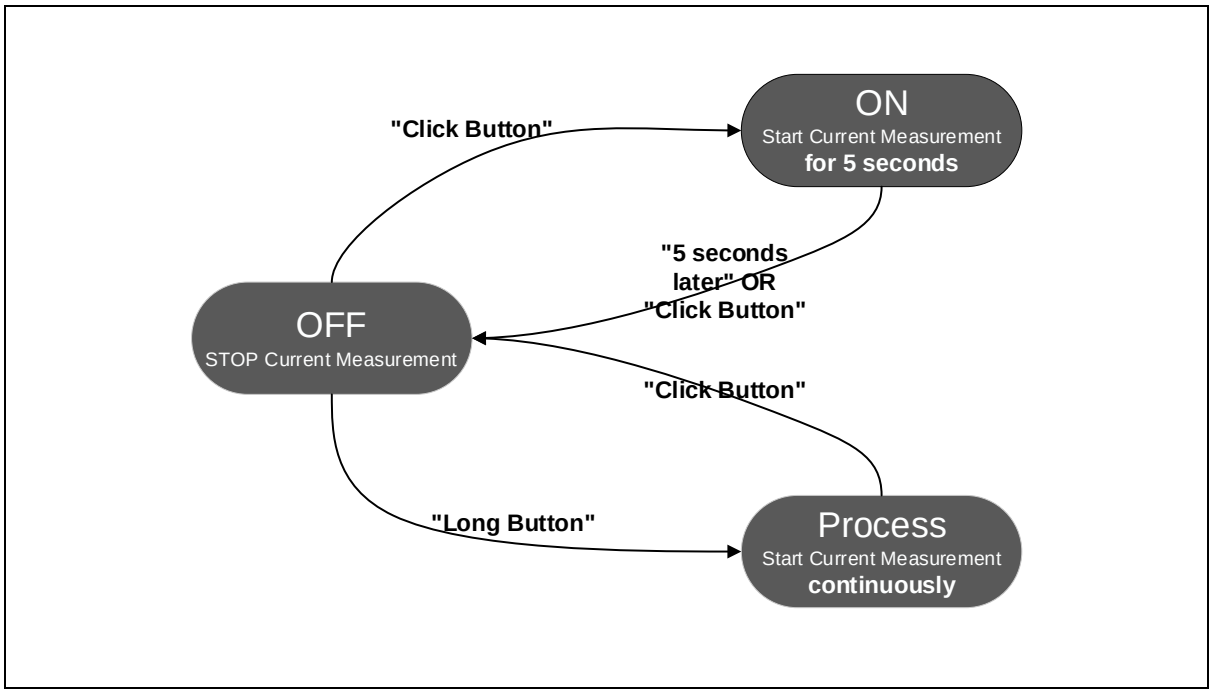


Figure 8. Current Measurement State Machine Diagram

4.1 State Machine Structure

Example software in this chapter is designed in a form of State Machine shown in Figure 8:

1. The State Machine consists of Function Pointers, and each of them points to executable code in each State.
2. An event that changes the current State is generated by a button.
3. The State Machine operates as follows:
   - At first on OFF State, if pressing the button, the current State changes to On State.
   - If pressing the button for a long period, the current State changes to Process State.
   - During On State, Current Sensing is executed for 5 seconds. If pressing the button within 5 seconds, the current State changes to Off State to stop the Current Sensing.
4. During Process State, the State Machine proceeds the Current Sensing and displays the measurement in real-time. If pressing the button, the measurement stops.
5. Users are free to modify the Structure of the State Machine, and add events.

4.2 State Machine Software Code

```
//*****
//* Function prototypes
//*****/
typedef void (* const voidFunc)(void);

voidFunc UponEnter[S_MAX] =        {State_Enter_OFF,   State_Enter_ON,   State_Enter_PROCESS};
voidFunc ActionWhileInState[S_MAX] = {State_InState_OFF, State_InState_ON, State_InState_PROCESS};
voidFunc UponExit[S_MAX] =         {State_Exit_OFF,    State_Exit_ON,    State_Exit_PROCESS};
```

Figure 9. Function Declaration using Function Pointers

Figure 9 shows an example code of function declarations that use Function Pointers.

1. UponEnter[S_MAX] includes functions executed when the current State changes (entering new State) due to the event occurrence such as button-pressing.
2. ActionWhileInState[S_MAX] includes functions executed after the UponEnter functions. Without special events, it continues to execute.
3. UponExit[S_MAX] includes functions executed when the current State changes (exiting the current State) due to the event occurrence such as button-pressing.
4. Users can modify the code by adding more functions.

```
enum states StateMachine(enum events event, enum states Current_State)
{
   int Next_State = Current_State;

   switch ( Current_State )
   {
   case S_OFF:
        switch (event )
        {
           // A transition to the next state will occur here
           case E_ON:
            Next_State = S_ON;
              break;
          case E_START:
            Next_State = S_PROCESS;
              break;
           default:  // Default case placed here to avoid Eclipse warnings as Eclipse expects
             break;  //to handle all enumerated values
        }
        break;
     case S_ON:
        switch (event )
        {
           // A transition to the next state will occur here
           case E_OFF:
            Next_State = S_OFF;
              break;
           default:  // Default case placed here to avoid Eclipse warnings as Eclipse expects
             break;  //to handle all enumerated values
        }
        break;
     case S_PROCESS:
        switch (event )
        {
           // A transition to the next state will occur here
         case E_OFF:
          Next_State = S_OFF;
            break;
         default:  // Default case placed here to avoid Eclipse warnings as Eclipse expects
          break; //to handle all enumerated values
        }
        break;
        // The program should never arrive here
    default:
        break;

   }
```

Figure 10. Example Code 1/2 showing StateMachine Function Structure

Figure 10 shows the first half of example code that is implemented in the StateMachine function. The example code operates as follows:

1. When the StateMachine function is called, the event and current State is stored as parameters.
2. During the current State, next State is set according to the event. For example, when the current State is S_OFF, if a button is pressed, an event = E_ON and Next_State = S_ON.
3. The event is classified by the length how long it is pressed.

```
    if (Next_State != Current_State)
    {
        // Function call for Upon Exit function, it can be omitted but allows extra functionality
        UponExit[Current_State]();
        // Function call for Upon Enter function, it can be omitted but allows extra functionality
        if (Next_State != S_MAX) UponEnter[Next_State]();
    }
    else // ActionWhileInState is only be called when NOT doing a transition
     {
        if ( event != E_MAX)
        {

        }
        else if(Current_State == S_PROCESS)
        {
          ActionWhileInState[Current_State]();
        }
        else if(Current_State == S_ON  && measure_count < 500)
        {
          ActionWhileInState[Current_State]();
          //3sec timer measure
        }
        else if(Current_State == S_ON  && measure_count >= 500)
        {
          input_key = 2;
          status_key = 2;

          Next_State = S_OFF;

          // Function call for Upon Exit function, it can be omitted but allows extra functionality
          UponExit[Current_State]();
          // Function call for Upon Enter function, it can be omitted but allows extra functionality
          if (Next_State != S_MAX) UponEnter[Next_State]();

          measure_count = 0;
          measure_key = 0;
        }
    }
    return (enum states)Next_State;
}
```

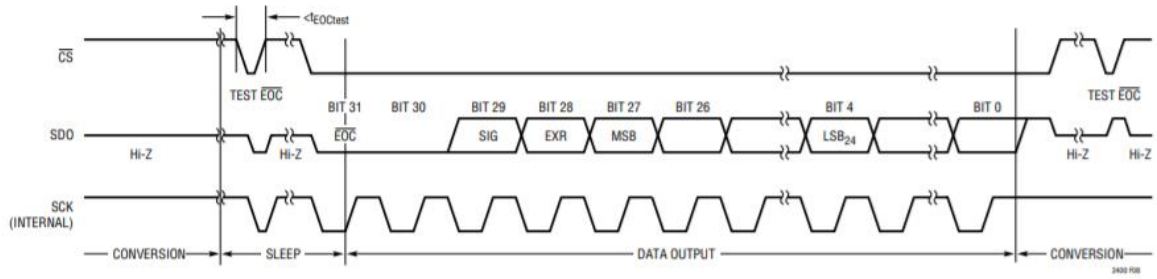
Figure 11. Example Code 2/2 showing StateMachine Function Structure

Figure 11 shows the second half of example code that is implemented in the StateMachine function. The example code operates as follows:

1. According to the current State and next State processed in the first half of the example code, functions to be called are determined.
2. If the current State and next State are different each other, a function in the UponExit of the current State is called and a function in the UponEnter of the next State is called in turn.
3. If the current State and next State are the same each other, functions in the ActionWhileInState of the current State are called continuously by default.
   - Example) The current State and next State are S_ON, functions in the ActionWhileInState (current measurement code) are executed for 5 seconds and terminated.
4. Users are free to modify the Structure of the State Machine, and add events.

4.3 24-bit ADC Communication Software Code

Figure 12 shows an example software code that intends to communicate with the 24-bit ADC shown in Figure 5.




```
/*---------------------------------------------------------------------------
 * @brief       External Interrupt 0 Handler
 * @return      none
 *//*-------------------------------------------------------------------------
void EINT0_Handler_IT( void )
{
  uint32_t    status;

  // get interrupt flag
  status = HAL_INT_EIntPA_GetIntStatus();

  // if interrupt was issued by PA5
  if( status & _BIT( 5 ) )
  {
    // clear interrupt flag
    HAL_INT_EIntPA_ClearIntStatus( _BIT( 5 ) );

    temp_val = HAL_GPIO_ReadPin( (Pn_Type* )PA );
    // output high
    if( (temp_val & (0x1<<7)) )
    {
      //HAL_GPIO_WritePin( ( Pn_Type* )PB, _BIT( 0 ) );
      spi_data |= (1<<(31-spi_data_cnt));
    }
    else
    {       // output low
      //HAL_GPIO_ClearPin( ( Pn_Type* )PB, _BIT( 0 ) );
    }

    spi_data_cnt++;
    if(spi_data_cnt >= 32)
    {
      // set flag
      convert_data = (0x1FFFFFF0 & spi_data);
      convert_data = (convert_data>>4);
      FlagPA0 = 1;
      //for debug mesg
      //_DBH32(convert_data);_DBG( "  <-  convert data finish₩n₩r" );

      spi_data_cnt = 0;
      spi_data = 0;
    }
  }
}
```

Figure 12. 24-bit ADC (LTC2400) SPI Comm. Operation Code and Waveforms

1. If the MCU sets the CS pin to LOW, the ADC outputs through the SCK and SDO (always to LOW).

2. To receive ADC 32-bit data consecutively, the level state of the SDO is read by using the GPIO external interrupt at Positive Edge (see the above example code in Figure 12).
3. When spi_data fills up the 32-bit length, BIT 4 to BIT 27 that are corresponding to the 24-bit ADC data is converted to 24-bit data by bit masking.

5 Quick Start Guide: Setup and Use

In Chapter 5, users can learn how to setup and use the Current Sensing Board that is designed using the example code of Chapter 4.

5.1 Current Sensing Evaluation Board

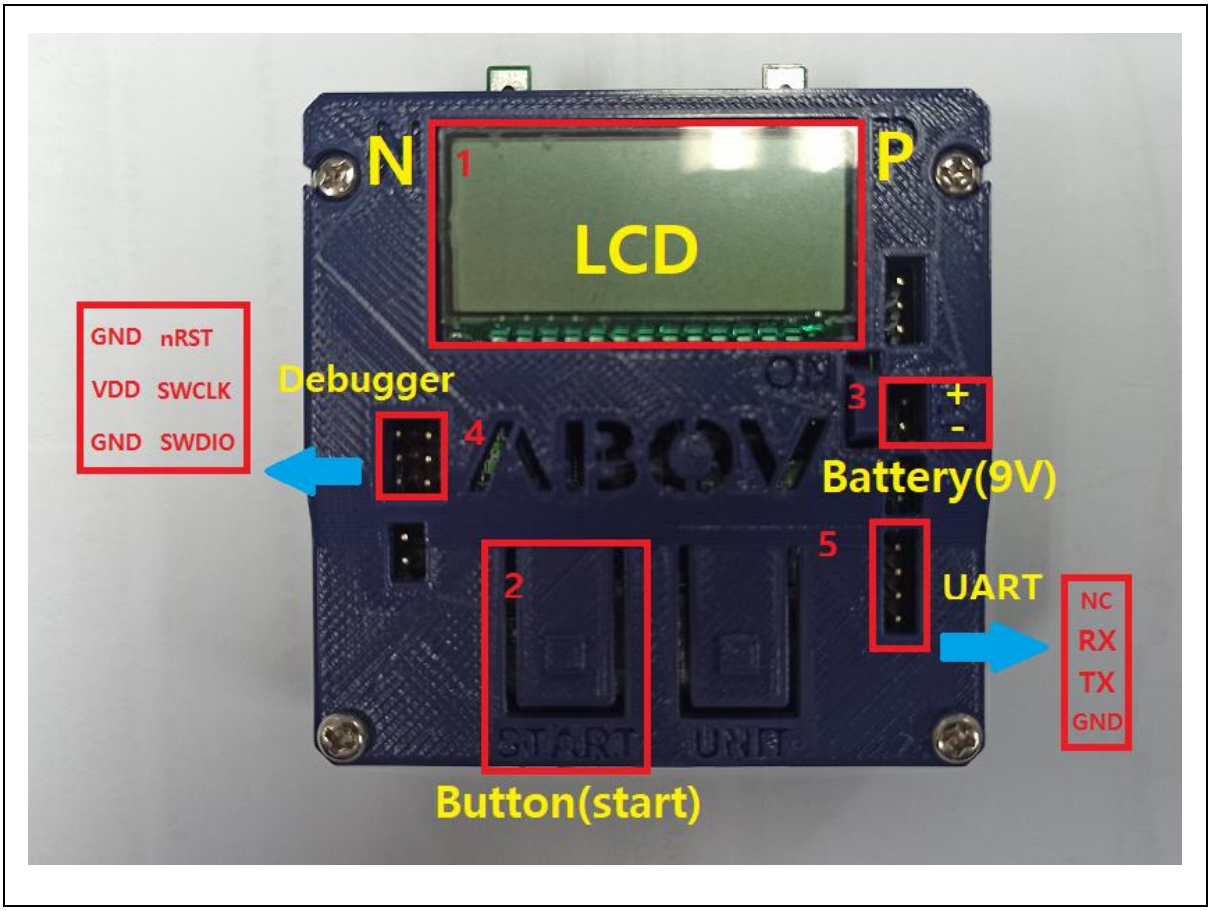


Figure 13. Current Sensing Evaluation Board

The hardware on the Current Sensing Evaluation Board shown in Figure 13 is as follows:

1. An LCD module and current sensing input terminals N and P
2. A button controlling the start and termination of the current measurement
3. An input terminal dedicated to a 9V voltage battery
4. A terminal for the MCU (A31L12X) download and debugging (SWD). Please refer to the pin map in red box next to Debugger terminal in Figure 13.
5. A terminal for MCU (A31L12X) UART communication. It is used for the debugging and test.

5.2 Example Firmware Project Compiling and Downloading

In this section, users can learn a method to compile, debug, and download an example firmware project.

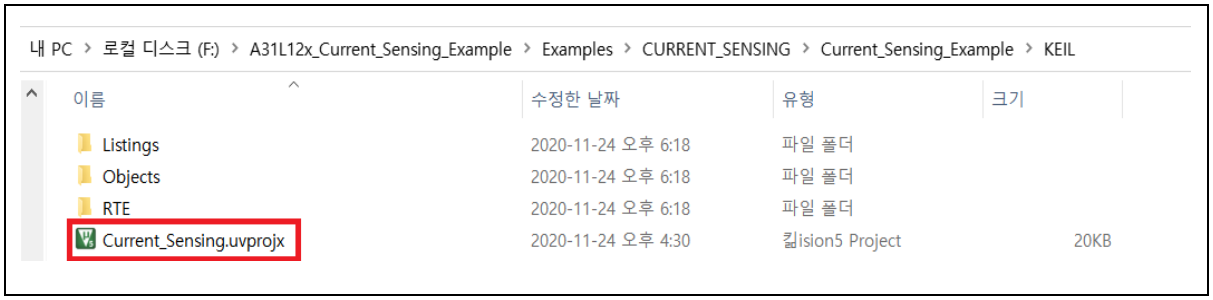


Figure 14. Example Project Folder Directory

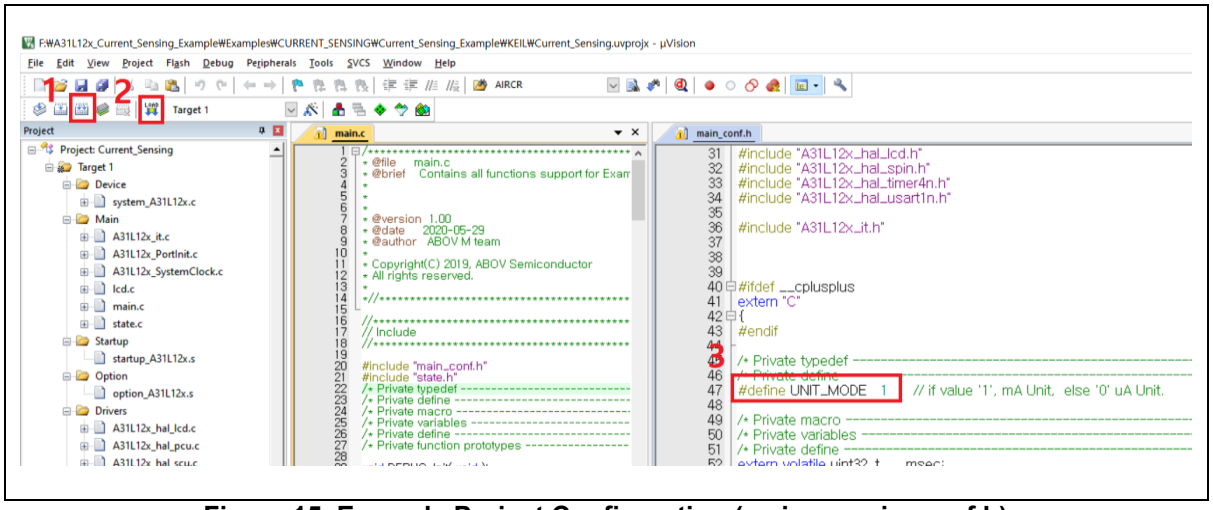


Figure 15. Example Project Configuration (main.c, main_conf.h)

Following is the description of the folder directory and project menus of the example project shown in Figure 14 and Figure 15:

1. Go to the location shown in Figure 14:
   A31L12x_Current_Sensing_Example\Examples\CURRENT_SENSING\
   Current_Sensing_Example\KEIL\

   ① Open the Current_Sensing.uvprojx file in the location.

   ② Execute Keil uVision.

2. Before starting the project compilation shown in Figure 15, set the UNIT_MODE on the 47th line of the main_conf.h file according to the measurement unit.

   — If the UNIT_MODE is set to 0, the measurement unit is uA while if the UNIT_MODE is set to 1, the measurement unit is mA.

3. Click on the icon 1 shown in Figure 15 to start compilation.
4. When the compilation is completed, connect to the debugger terminal on the example board.
5. Click on the icon 2 shown in Figure 15 to start firmware download.
6. Information of the debugger terminal is available in **5.1 Current Sensing Evaluation Board**.

5.3 How to use Current Measurement Board

This section explains a method to measure the current by using the Current Sensing Board.

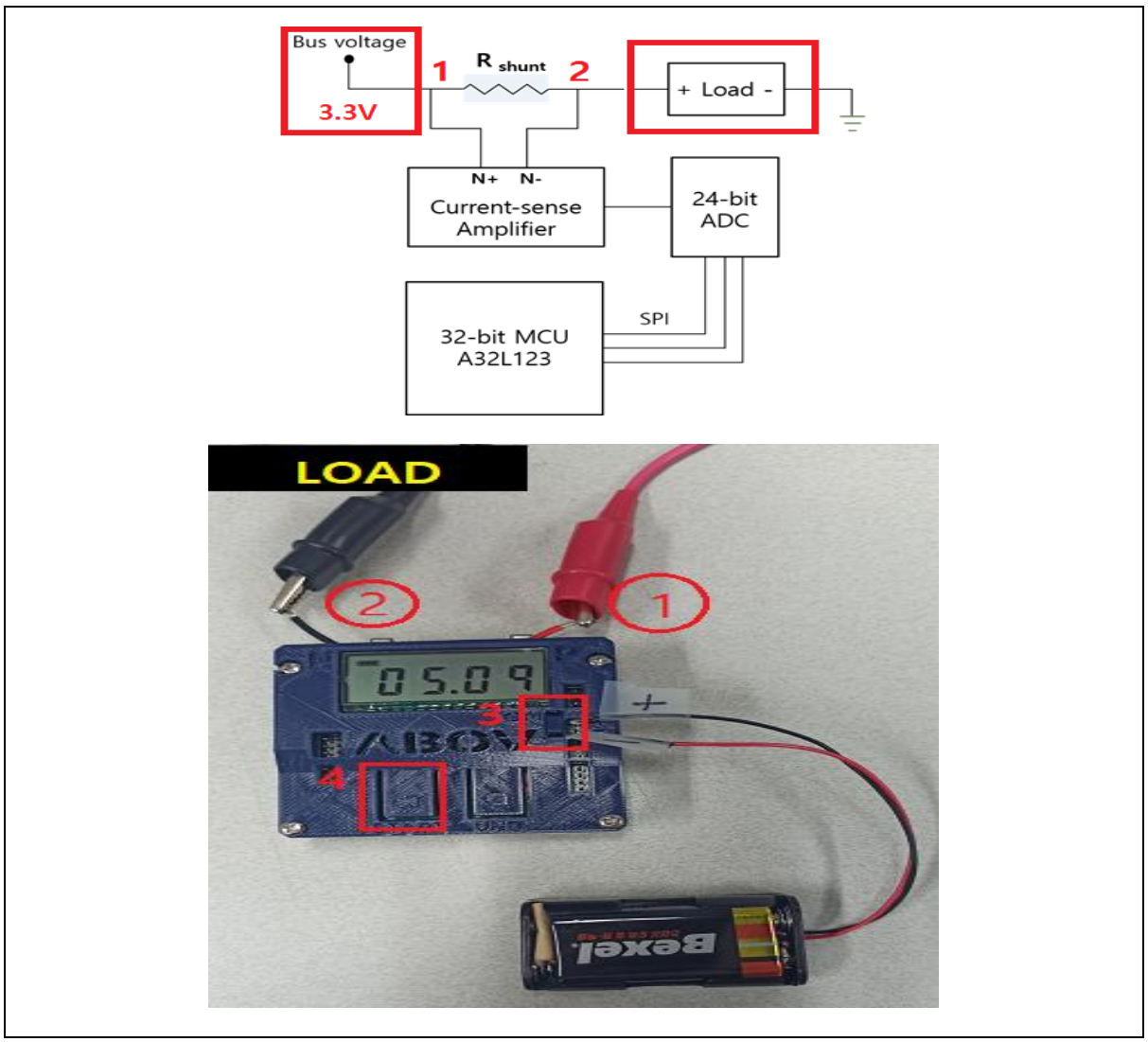


Figure 16. Current Sensing Board Connections

Connections of the Current Sensing Board shown in Figure 16 are detailed in the followings:

1. Connect the board as shown in Figure 16 and turn on the power of the Current Sensing Board.
2. Identically to the current measurement mode of multimeter, connect the power of the target board to N+ and N- of measuring meter in serial.
3. When the power is applied to N+(1), the power of the target board inflows through N-(2) and the current is measured.

4. Press the button 4 shown in Figure 16 to start the current measurement.
   - If pressing the button for a long period, it means to proceed the current measurement continuously.
   - If pressing the button for short period, it means to terminate the current measurement in 5seconds.
5. The measured value is calculated to the decimal point second digit and displayed on the LCD.

5.4 Example Circuit Diagram and Firmware Source for Current Sensing Board

Figure 17 and Figure 18 show the example circuit diagrams of the Current Sensing Board.

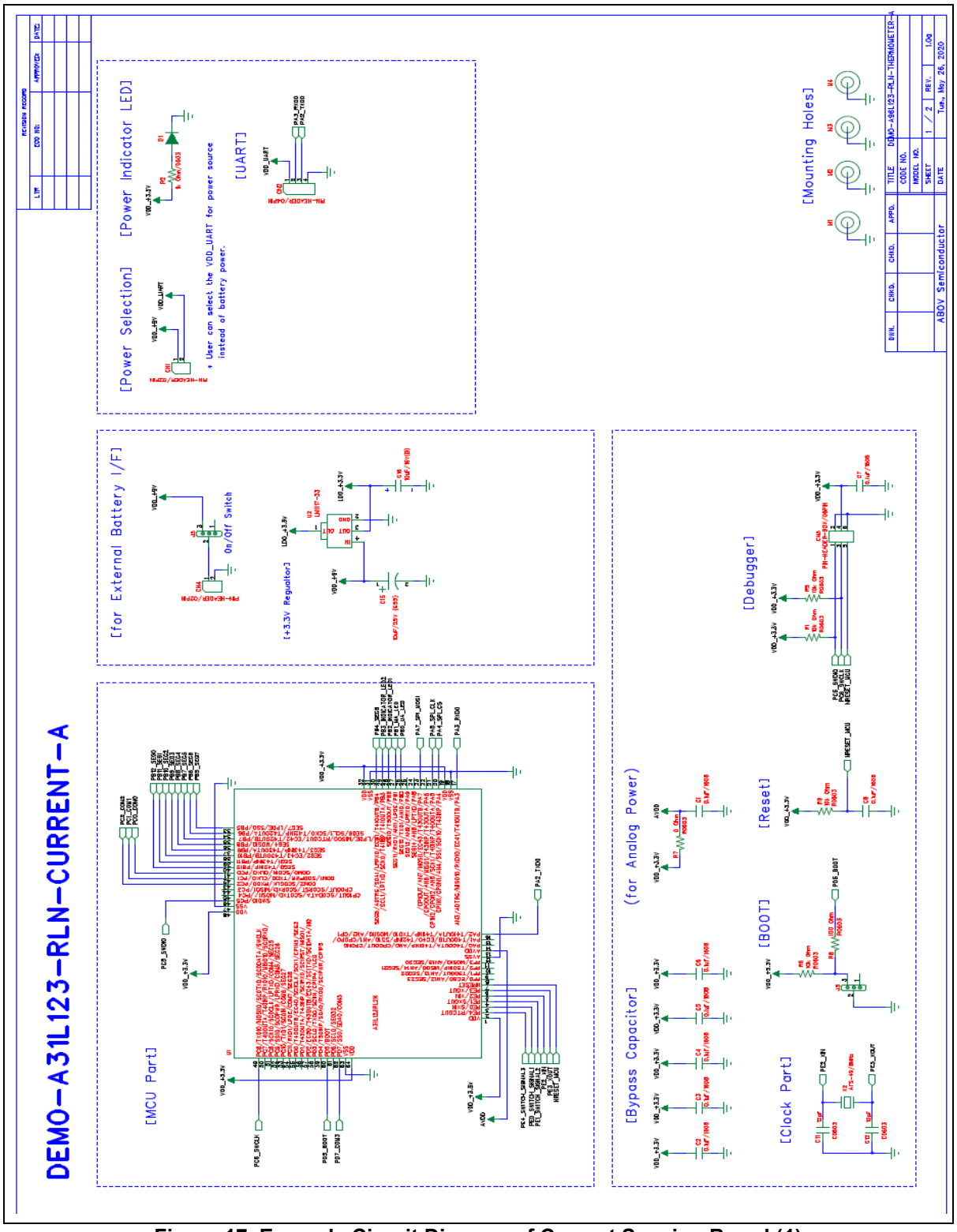


Figure 17. Example Circuit Diagram of Current Sensing Board (1)

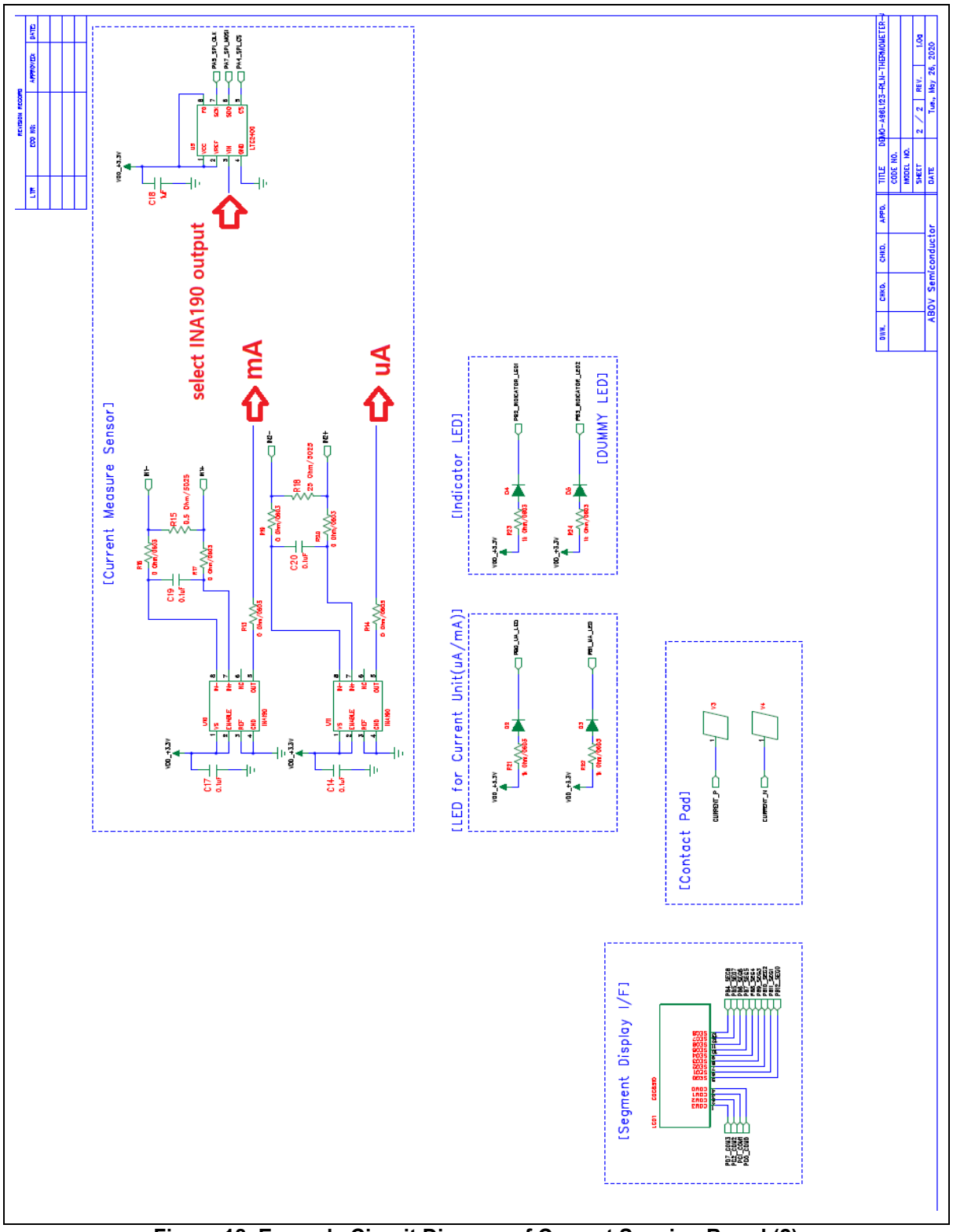


Figure 18. Example Circuit Diagram of Current Sensing Board (2)

By providing the example source on ABOV's website, users can open the example firmware and utilize it in actual design.

6 Conclusion

This document introduces a method to design hardware and configure software for the current sensing and measurement. This document provides actual example board to help users design and use the hardware quickly.

7 References

- Texas Instrument — INA190 (Current-Sense Amplifier)
- Linear Technology — LCT2400 (24-bit ADC)
- ABOV Semi — A31L12X (32bit-MCU)

Revision History

Version	Date	Description
1.00	20.11.30	Initial preliminary version created
1.01	22.11.14	Revised the font of this document
1.02	24.12.02	Updated the disclaimer.

Korea

Regional Office, Seoul
R&D, Marketing & Sales
8th Fl., 330, Yeongdong-daero,
Gangnam-gu, Seoul,
06177, Korea

Tel: +82-2-2193-2200
Fax: +82-2-508-6903
www.abovsemi.com

HQ, Ochang
R&D, QA, and Test Center
37, Gangni 1-gil, Ochang-eup,
Cheongwon-gun,
Chungcheongbuk-do,
28126, Korea

Tel: +82-43-219-5200
Fax: +82-43-217-3534
https://www.abovsemi.com/en/main.php

Domestic Sales Manager
Tel: +82-2-2193-2206
Fax: +82-2-508-6903
Email: sales_kr@abov.co.kr

Global Sales Manager
Tel: +82-2-2193-2281
Fax: +82-2-508-6903
Email: sales_gl@abov.co.kr

China Sales Manager
Tel: +86-755-8287-2205
Fax: +86-755-8287-2204
Email: sales_cn@abov.co.kr