**INTRODUCTION**

The BCB606 demonstration board provides a hardware platform and firmware for interacting with the 5-channel automotive PMIC Stowe in a way that is substantially similar, and largely compatible, with ADI’s internal bench evaluation system. It is a baseboard, or motherboard, that receives a target board for the Stowe PMIC IC (aka LT3390 / LT3390-3 / LT3390-5 /…). A target board for each supported variant of Stowe is available from ADI. For custom applications of Stowe, please contact your ADI sales or applications representative.

The BCB606 also provides the services needed to interact with other target-board resources such as a precision temperature sensor and an EEPROM for electronically identifying the target-board.

The demo-board incorporates a ATSAMD21 microcontroller programmed to emulate an Arduino M0 (Native USB) so that firmware experimentation can be performed with as little setup burden as possible. It also includes an LTM2884 USB isolator module so that board faults, such as power supply reversal, don’t damage the host computer.

The BCB606 is designed to interface with a host computer via the Labcomm protocol. Labcomm is a lightweight binary frame packet protocol that can be transmitted over USB. A Labcomm parser and a Code-Free GUI™ are available from [PyICe on GitHub](https://github.com/Xenomorphxx121/PyICe). PyICe is Analog Device’s **Py**thon **I**ntegrated **C**ircuit **E**nvironment, a Python library for holistically interacting with integrated circuits and laboratory test equipment. Contact your Analog Devices applications engineer for assistance in installing PyICe on your computer. Below is an image of the PyICe Code-Free GUI™ running on the BCB606:

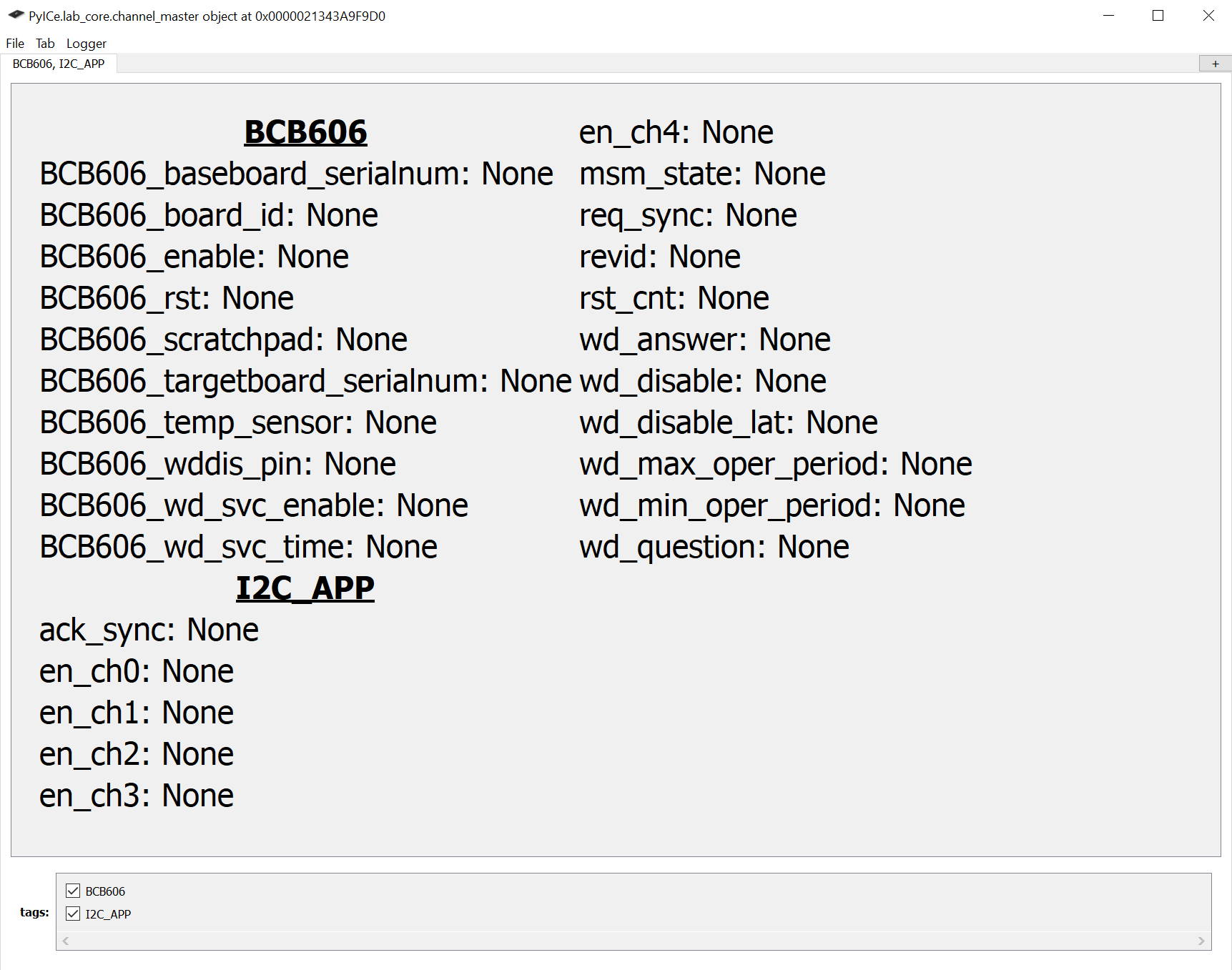


Figure 1.The PyICe Code-Free GUI

The BCB606 firmware stack consists of the following modules, each of which will be described below.

**BCB606.ino**

**BCB606\_heartbeat.cpp**

**BCB606\_enable\_pin.cpp**

**BCB606\_identify.cpp**

**BCB606\_rst\_pin.cpp**

**BCB606\_wddis\_pin.cpp**

**BCB606\_watchdog.cpp**

**BCB606\_postoffice.cpp**

**BCB606\_eeprom.cpp**

**BCB606\_smbus.cpp**

**BCB606\_smbus\_services.cpp**

**BCB606\_softport.cpp**

**BCB606\_temp\_sensor.cpp**

**labcomm.cpp**

**stowe\_pec.cpp**

**Depdendencies**

The Arduino project has the following dependencies:

**• FlashStorage\_SAMD**

**• SoftWire**

**• AsyncDelay**

**BCB606.ino**

This main program file contains the hardware setup routine and main program loop. The main program consists of a simple round-robin operating system that calls on the available services repeatedly with no specific priority. Each service will generally only perform a function if there a Labcomm message requesting it do so but will more likely do nothing and drop straight through. One exception to this is the watchdog service routine which will service Stowe’s question-answer watchdog if enabled to do so via a Labcomm command from the USB port.

**BCB606\_postoffice.cpp**

The post office creates ***Mailbox*** objects and executes simple send and receive methods. Each of the mailboxes consist of a data structure and methods for receiving and dispatching messages via Labcomm. The post office is custom to the BCB606 because the endpoints, or message recipients, are custom. The post office was designed to be as easily comprehensible and extensible as possible for re-use on other Labcomm projects. The **Mailbox** class is defined in labcomm.h.

**BCB606\_heartbeat.cpp**

The heartbeat routine reads the Arduino’s millisecond clock and enables an LED in a heartbeat-like pattern once per second. If the micro-controller is unburdened, the heartbeat will look quite regular, i.e., heartbeat like. Conversely, if there is a high burden to send or receive voluminous SMBus data, the heartbeat will not be serviced regularly and an irregular beating pattern will appear.

**BCB606\_enable\_pin.cpp**

The enable pin endpoint can set the Stowe ENABLE pin to low, high or test-hook. When low, the Stowe PMIC is off and SMBus communication should not be expected. When high, Stowe will be on, it will communicate and provide output voltages. When transitioned from low to test-hook, Stowe will enter its service mode. Service mode requires the use of a 12V, center positive, wall adapter with a 5.5mm barrel jack (supplied with the BCB606) at connector J1 and should only be used with advice and consent of factory trained Analog Devices applications or design engineers.

The enable pin module has a simple Labcomm payload command structure consisting of 2-bytes in and 1-byte out.

The Enable pin’s packet payload 2-byte structure is:

|  |  |
| --- | --- |
| COMMAND  (1 Byte) | DATA  (1 Byte) |

Table . Enable Pin Module Command Structure

The ENABLE pin’s command values are as follows:

|  |  |
| --- | --- |
| SET\_STATE | 1 |
| GET\_STATE | 2 |

Table . Enable Pin Module Commands

The SET\_STATE data values are:

|  |  |
| --- | --- |
| DISABLE\_STOWE | 0 |
| ENABLE\_STOWE | 1 |
| TEST\_HOOK\_STOWE | 2 |

Table . Enable Pin Settings

When the GET\_STATE command is deployed, a single byte is returned with the current state of the ENABLE pin from Table3.

**BCB606\_identify.cpp**

This module returns identifying information about the firmware version currently running on the BCB606. It has a simple one-byte command structure. To retrieve the board’s version information, the payload should contain the ASCII value for the simple question mark “**?**”.

The identify module also has the ability to set and retrieve a 255 byte nonvolatile scratchpad value that can be used to leave notes or declare ownership, for instance. There is also a unique serial number for each instance of the BCB606. The serial number is retrieved from the 128-bit (16 byte) memory location that is factory programmed in the SAMD21 microcontroller. The commands to set and get the scratchpad are 1 and 2 respectively. The 128-bit serial number is retreived with the command 3. The table below shows the expected command and data fields for each command.

|  |  |  |
| --- | --- | --- |
| **Command** | **Code** | **Data** |
| Identify | "?" (63 or 0x3F) | Returns ID String (of Labcomm payload size) |
| Write Scratchpad | 1 | Send up to 255 Bytes (Chars) |
| Read Scratchpad | 2 | Expect back up to 255 Bytes (Chars) |
| Get Serial Number | 3 | Reads back the Arduino M0’s (SAMD21) unique 128 bit (16 bytes) internal ID |

Table . Identify Module Commands

**BCB606\_wddis\_pin.cpp**

Stowe has a WD\_DISABLE pin for either, eliminating the watchdog feature, or for temporarily disabling it for product debug and development. The WD\_DISABLE pin has a 2-byte command structure.

The WD\_DISABLE pin’s packet payload 2-byte structure is:

|  |  |
| --- | --- |
| COMMAND  (1 Byte) | DATA  (1 Byte) |

Table . WD\_DISABLE Pin Module Command Structure

The WD\_DISABLE pin’s command values are as follows:

|  |  |
| --- | --- |
| SET\_STATE | 1 |
| GET\_STATE | 2 |

Table . WD\_DISABLE Module Commands

The SET\_STATE data values are:

|  |  |
| --- | --- |
| DISABLE\_WDDIS | 0 |
| ENABLE\_WDDIS | 1 |

Table . WD\_DISABLE Pin Commands

When the GET\_STATE command is deployed, a single byte with the current state of the WD\_DISABLE pin from Table 6 is returned.

**BCB606\_watchdog.cpp**

The watchdog routine shows an example of how to service Stowe’s watchdog function. It periodically checks the system’s micro-second clock and decides if the watchdog should be serviced.

The LT3390 Stowe variant has a minimum open window time of 16ms and a maximum open window time of 144ms. The default BCB606 time has been pre-programmed at 48ms to land geometrically between the open and closed window values. Its value can be adjusted remotely as shown in Table 7.

The payload for the watchdog module expects one to five bytes in and returns 4-bytes out.

The Watchdog Module command structure is as follows:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Command** | **Code** | **Data** |  |  |  |
| SET\_SERVICE STATE | 0 | 0/1 | -- | -- | -- |
| GET\_SERVICE\_STATE | 1 | -- | -- | -- | -- |
| SET\_RESPONSE\_TIME\_us | 2 | MS-Byte | Byte3 | Byte2 | LS-Byte |
| GET\_RESPONSE\_TIME\_us | 3 | -- | -- | -- | -- |
| SET\_ADDR7 | 4 | Addr7 | -- | -- | -- |
| GET\_ADDR7 | 5 | -- | -- | -- | -- |
| SET\_SERVICE\_METHOD | 6 | 0/1 | -- | -- | -- |
| GET\_SERVICE\_METHOD | 7 | -- | -- | -- | -- |
| SET\_USE\_PEC | 8 | 0/1 | -- | -- | -- |
| GET\_USE\_PEC | 9 | -- | -- | -- | -- |
| SET\_QUESTION\_ADDR | 10 | QUESTION ADDR | -- | -- | -- |
| GET\_QUESTION\_ADDR | 11 | -- | -- | -- | -- |
| SET\_ANSWER\_ADDR | 12 | ANS\_ADDR | -- | -- | -- |
| GET\_ANSWER\_ADDR | 13 | -- | -- | -- | -- |
| SET\_CRC\_POLY | 14 | POLY | -- | -- | -- |
| GET\_CRC\_POLY | 15 | -- | -- | -- | -- |

SET\_SERVICE\_STATETable Watchdog Module Command Structure and Settings

The response time is a 4-byte (32-bit integer) value representing the number of micro-seconds from one successful service of a watchdog to the next. Depending on loading from the other services on the demo-board, and the operating system, the service time will not be exactly this value. It will always be longer but will never be shorter.

The return value from the Get\_Response\_Time query is a similar 4-byte (32 bit) integer representing the micro-seconds dwell to which the watchdog routine is currently set. The 4-byte, 32 bit, number is consistent with the Arduino’s micros() function from the ***Arduino.h*** library.

Two methods of generating watchdog replies are demonstrated with this firmware stack, lookup table and algorithmic. The look up table, which can be stored in abundant program memory is faster but uses a bit pf program space. The algorithmic method uses little memory but takes a bit more execution time. Note that changing the method from the user interface will give no outward appearance of a change. The option is only provided as a code example for customers to duplicate.

The GET\_SERVICE\_METHOD values are:

|  |  |
| --- | --- |
| USE\_LOOKUP\_TABLE | 0 |
| USE\_ALGORITHMIC | 1 |

There are also options to alter the question address, answer address, CRC polynomial, use of PEC, etc.

**BCB606\_rst\_pin.cpp**

The RST pin module expects no incoming payload (payload size=0) and responds with a single byte written in response. Merely addressing the RST module with no payload is sufficient to trigger the module to return Stowe’s RST pin value to the host. The one-byte response will be 1 if RST is high and 0 if it is low.

**BCB606\_eeprom.cpp**

**BCB606\_smbus\_services.cpp**

**BCB606\_temp\_sensor.cpp**

These modules are substantially pass-through modules that take higher level commands from a PyICe controlled host and relays them to the SMBus two-wire interface modules *BCB606\_smbus.cpp* and *BCB606\_softport.cpp*.

BCB606\_smbus\_services.cpp is the module that interacts with the Stowe SMBus (I2C) port and, as such, it offers additional services. Specifically, it has the 5-byte payload command structure shown below followed by a data-list.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **COMMAND** | **ADDR7** | **COMMAND CODE** | **USE PEC** | **DATASIZE (8/16)** | **REG1** | **REG2** | **REG3** | **...** |

Table . SMBus Module Command Structure

Where the transaction type or command table is:

|  |  |  |  |
| --- | --- | --- | --- |
| **Command** | **Code** | **Bytes Sent**  **(Byte/Word)** | **Bytes Returned**  **(Byte/Word)** |
| SMBUS\_QUICK\_COMAND | 1 | 0 | 0 |
| SMBUS\_SEND\_BYTE | 2 | 1 | 0 |
| SMBUS\_RECEIVE\_BYTE | 3 | 0 | 1 |
| SMBUS\_WRITE\_REGISTER | 4 | 1/2 | 0 |
| SMBUS\_READ\_REGISTER | 5 | 0 | 1/2 |
| SMBUS\_PROCESS\_CALL | 6 | X | 0 |
| SMBUS\_BLOCK\_WRITE | 7 | X | Unimplemented |
| SMBUS\_BLOCK\_READ | 8 | X | Unimplemented |
| SET\_REGISTER\_LIST | 21 | #bytes/2x#words | 256/512 |
| READ\_REGISTER\_LIST | 22 | #bytes/2x#words | 256/512 |
| ENABLE\_STREAM\_MODE | 23 | X | 0 |
| DISABLE\_STREAM\_MODE | 24 | X | 0 |
| WRITE\_REGISTER\_LIST | 25 | #bytes/2x#words | 512 ([REG,VAL] Pairs) |
| SET\_REG\_LIST\_AND\_READ\_LIST | 26 | 8 | 256 |
| SET\_REG\_LIST\_AND\_STREAM | 27 | 8 | 256 |

Table . SMBus Module Commands and Expectant Data Sizes

Every SMBus transaction returns a status byte to the SMBus post office box with which is communicating. A return value of 0 indicates that the firmware was satisfied with the transaction whereas as a nonzero value implies an issue. Please see the 1-Hot table below:

|  |  |
| --- | --- |
| **Return Code** | **Status** |
| 0 | SMBUS\_SUCCESS |
| 1 | SMBUS\_NACK\_ON\_ADDRESS |
| 2 | SMBUS\_NACK\_ON\_DATA |
| 4 | SMBUS\_PEC\_VALUE\_ERROR |
| 8 | SMBUS\_SMBUS\_TIMEOUT |
| 16 | SMBUS\_BUFFER\_OVERFLOW |
| 32 | SMBUS\_UNKNOWN\_ERROR |

*Table 11. SMBus Module Return Values*

**BCB606\_eeprom.cpp**

Because Stowe has many variants, each target board comes electronically marked with an EEPROM (BR24H64FVM-5AC) for identification. Target boards may also be customized by ADI applications engineers for your application. A serial number in the EEPROM helps ADI track and debug your target board and application more readily. There is a target board registry at Analog Devices. If you have questions about how your target board is configured, please contact you ADI applications or design engineer for assistance with your specific application and target board.

**BCB606\_temp\_sensor.cpp**

The Stowe target board uses a temperature sensor to monitor the ambient temperature. This can be useful for laboratory evaluation when using an environmental temperature chamber. Be aware that the BCB606, itself, may not be suitable for use over a wide operating temperature range. The Stowe development team has a different motherboard intended specifically for use in a wide-range temperature chamber. To get Stowe characterization data over temperature, please contact your local ADI sales office or applications engineer.

**BCB606\_smbus.cpp**

**BCB606\_softport.cpp**

**stowe\_pec.cpp**

These are ancillary modules that provide low level commands for communicating with Stowe and the other I2C/SMBus peripherals on the target board.

The BCB606\_smbus.cpp module accesses the built in two-wire hardware of the ATSAMD21 microcontroller and is used for communicating with the Stowe PMIC.

The stowe\_pec.cpp module contains routines that validate the SMBus PEC values.

The BCB606\_softport.cpp module forms a two-wire master out of two standard GPIO pins on the microcontroller for a slower, but independent, secondary SMBus port. It follows the same 5-byte command structure as the BCB606\_smbus\_services.cpp as shown in Table 9 and is used to communicate with the EEPROM and temperature sensor on the target board.