# Intel® AtomTM Apollo Lake Processor Windows 10 IO Driver Software Developer's Manual

---

*September 2018*

Document Number:

# Contents

# Figures

No table of figures entries found.

# Tables

**No table of figures entries found.**

# *1.0 CHAPTER 1: About this Manual*

## 1.1 Operating System Covered in this Manual

This Manual set includes information pertaining to the following set of Operating system

- Windows 10 IOT Enterprise 64 Bit RS4
- Windows 10 IOT Core 64 Bit RS4

The IO drivers are dependent on the Operating System (OS) driver installation.

# 2.0 CHAPTER 2 General Purpose Input Output (GPIO) Driver

This section provides the programming details and interface exposed by General Purpose Input Output (GPIO) driver for Windows 10. The current implementation of the driver exposes the interfaces through Input / Output Controls (IOCTLs), which can be called from the application (user mode) using the API DeviceIoControl (Refer to the MSDN documentation for more details on this API). The following sections provide information about the IOCTL and how to use them to configure the GPIO hardware.

## 2.1 Driver Features

The GPIO Driver Supports:

- Writing data to GPIO hardware
- Reading data from GPIO hardware
- Setting the direction of GPIO hardware

**Limitation:**

- Multiplexing is not supported. Any pin-multiplexing and pad configuration must be completed in the firmware prior to handing control to the OS loader.

## 2.2 Interface Details

Table below lists IOCTLs supported by the driver

| No | IOCTL | Remark |
|---|---|---|
| 1 | IOCTL_GPOT_OPEN_INPUT | Set as input direction of selected pin of given GPIO controller |
| 2 | IOCTL_GPOT_OPEN_OUTPUT | Set as output direction of selected pin of given GPIO controller |
| 3 | IOCTL_GPOT_READ | To read from a set of GPIO pins that are configured as inputs. |
| 4 | IOCTL_GPOT_CLOSE | Close IO pin and set to default |
| 5 | IOCTL_GPOT_GET_STATUS | Read the pin status of selected pin of given GPIO controller |
| 6 | IOCTL_GPOT_GET_RESOURCE_COUNT | Count the pin resource of GPIO |
| 7 | IOCTL_GPOT_WRITE_HIGH / LOW | To write to a set of GPIO pins that are configured as outputs |

## 2.3 IOCTL Usage Details

This section assumes a single client model where there is a single application-level program configuring the GPIO interface and initiating I/O operations. The following files contain the details of the IOCTLs and data structures used.

### 2.3.1 IOCTL_GPOT_OPEN_INPUT

This IOCTL is called to set the input direction of selected pin of given GPIO controller. The prerequisite is the device must be installed and opened using API CreateFile and the pin is set to GPIO function mode.

```
GPIO_PIN_PARAMETERS parameter;
Parameter.pin = pin;
DeviceIoControl (hHandle,
                IOCTL_GPOT_OPEN_INPUT,
                &parameter,
                sizeof (GPIO_PIN_PARAMETERS), NULL, 0,
                &dwSize,
                NULL);
```

### 2.3.2 IOCTL_GPOT_OPEN_OUTPUT

This IOCTL is called to set the output direction of selected pin of given GPIO controller. The prerequisite is the device must be installed and opened using API CreateFile and the pin is set to GPIO function mode.

```
GPIO_PIN_PARAMETERS parameter;
Parameter.pin = pin;
DeviceIoControl (hHandle,
                IOCTL_GPOT_OPEN_INPUT,
                &parameter,
                sizeof (GPIO_PIN_PARAMETERS), NULL, 0,
                &dwSize,
                NULL);
```

### 2.3.3 IOCTL_GPOT_CLOSE

This IOCTL is to close GPIO pin and set GPIO pin value back to default. Once all the operations related to GPIO driver are finished, the device handle must free the application by calling the IOCTL_GPOT_CLOSE.

```
GPIO_PIN_PARAMETERS parameter;
Parameter.pin = pin;
DeviceIoControl (CreateFile,
                IOCTL_GPOT_CLOSE,
                &parameter,
                sizeof (GPIO_PIN_PARAMETERS), NULL, 0,
                &dwSize,
                NULL);
```

### 2.3.4 IOCTL_GPOT_READ

This IOCTL is called to read from a set of GPIO pins that are configured as inputs. The prerequisite is the device must be installed and opened using API CreateFile. To read a value to a pin, the pin must first set to Input mode, refer to Section 2.3.1.

```
GPIO_PIN_PARAMETERS parameter;
Parameter.pin = pin;
GPIO_PIN_DATA pindata;
UNIT pindata = 0;
DeviceIoControl (hHandle,
                IOCTL_GPOT_READ,
                &parameter,
                sizeof (GPIO_PIN_PARAMETERS),
                &pindata,
                sizeof (GPIO_PIN_DATA),
                &dwSize,
                NULL);
```

### 2.3.5 IOCTL_GPOT_GET_STATUS

This IOCTL is to read the pin status from GPIO controller. The prerequisite is the device must be installed and opened using API CreateFile.

```
GPIO_PIN_PARAMETERS parameter;
Parameter.pin = pin;
GPOT_STATUS_ENTRY *pins;
DeviceIoControl (hHandle,
                IOCTL_GPOT_GET_STATUS, NULL, 0,
                &parameter,
                sizeof (GPIO_STATUS_ENTRY),
                &dwSize
                NULL);
```

### 2.3.6 IOCTL_GPOT_GET_RESOURCE_COUNT

This IOCTL is to request a controller-specific device-control operation and count the GPIO test pin.

```
GPIO_PIN_PARAMETERS parameter;
Parameter.pin = pin;
UNIT pcnt;
OVERLAPPED Overlapped;
DeviceIoControl (hHandle,
                IOCTL_GPOT_GET_RESOURCE_COUNT, NULL, 0,
                &pcnt,
                &dwSize ,
                &Overlapped)
```

### 2.3.7 IOCTL_GPOT_WRITE_HIGH / LOW

This IOCTL is to write to a set of GPIO pins that are configured as outputs. The value can be high or low. The prerequisite is the device must be installed and opened using API CreateFile. To write a value to pin, the pin must first set to output mode, refer to 2.3.2 section.

```
GPIO_PIN_PARAMETERS parameter;
Parameter.pin = pin;
DeviceIoControl (CreateFile,
                ((value) ? IOCTL_GPOT_WRITE_HIGH: IOCTL_GPOT_WRITE_LOW),
                &parameter,
                sizeof (GPIO_PIN_PARAMETERS), NULL, 0 ,
                &dwSize
                NULL);
```

## 2.4 Structures

All structures used by the interfaces are defined in public.h. Below is the structure used for preserving information related to the GPIO request.

Name

```
enum _GPOT_OPEN_STATE
{
     OpenStateNotOpened = 0,
     OpenStateInput,
     OpenStateOutput
} GPOT_OPEN_STATE;
```

```
struct _GPOT_STATUS_ENTRY
{
     LARGE_INTEGER ConnectionId;
     GPOT_OPEN_STATE OpenState;
     ULONG CFG0;
     ULONG CFG1;
} GPOT_STATUS_ENTRY;
```

## 2.5 Error Handling

Since the IOCTL command is implemented using the Windows API, the return value of the call is dependent on and defined by the OS. On Windows, the return value is a non-zero value. If the error is detected within or outside the driver, an appropriate system defined value is returned by the driver.

## 2.6 Inter-IOCTL Dependencies

There are no inter-IOCTL dependencies for GPIO driver. Once the driver is loaded successfully, the IOCTLs stated above can be used in any order.

## 2.7 Programming Guide for GPIO Driver

This section describes the basic procedure for using the GPIO driver from a user mode application. All operations are through the IOCTLs exposed by the GPIO driver. The steps involved in accessing the GPIO driver from the user mode application are described below:

- Open the device
- Initialize and configure the driver with desired settings through the interfaces exposed
- Perform read/write operations
- Close the device

# 3.0 CHAPTER 3: High Speed Universal Asynchronous Receiver-Transmitter (HSUART)

This section provides the programming details of the High Speed Universal Asynchronous Receiver-Transmitter (HSUART) driver for Windows 10. This included the information about the interfaces exposed by the driver and how to use the interfaces to driver the HSUART hardware through Input/Output Controls (IOCTLs), which can be called from the application (user mode) using the API DeviceIoControl. Refer to the MSDN documentation for more details on this API.

HSUART is an individual integrated circuit (IC) used for serial communication and a device for asynchronous serial communication in which the data format and transmission speeds are configurable.

## 3.1 Driver Features

The HSUART driver supports:

- Supports none, odd and even parity
- Supports data sizes of 5, 6, 7 and 8 bit
- Supports 1, 1.5 and 2 stop bits
- Support “Hardware” (RTS / CTS) and “No” flow control
- Supports baud rates of 300-921600, up to 3686400 by default. To set baud rates of 1M, 2M, 3M and 4M, refer to the Software Driver BKMs.
- Supports Serial Device Control Requests (IOCTLs) defined by Microsoft for serial controllers in Windows. Refer to the following Limitations section for IOCTLs

**Limitations:**

- When using 1.5 stop bits, the data size can only be supported up to 5 bits.
- DTR/DSR handshake is not supported
- The following are IOCTLs that are not supported in the driver:
    - IOCTL_SERIAL_XOFF_COUNTER
    - IOCTL_SERIAL_LSRMST_INSERT
    - IOCTL_SERIAL_SET_BREAK_ON
    - IOCTL_SERIAL_SET_BREAK_OFF

## 3.2 Interface Details

In Windows 10, to control a peripheral device that connected to a port on a serial controller, it uses IRP_MJ_WRITE and IRP_MJ_READ requests to transmit and receive data from a serial port. Windows defines a set of serial I/O control request (IOCTLs) that a client can use to configure a serial port.

Table below lists IOCTLs supported by the driver

| No | IOCTL | Remark |
|---|---|---|
| 1 | IOCTL_UARTTESTTOOL_OPEN | Enable and Open Multi-Com Port |
| 2 | IOCTL_SERIAL_SET_BAUD_RATE | Request set the baud rate on a serial controller device |
| 3 | IOCTL_SERIAL_GET_BAUD_RATE | Request returns the baud rate at which the serial port is currently configured to transmit and receive data |
| 4 | IOCTL_SERIAL_SET_MODEM_CONTROL | To set the modem control register |
| 5 | IOCTL_SERIAL_GET_MODEM_CONTROL | To obtain the value of the MCR |
| 6 | IOCTL_SERIAL_SET_LINE_CONTROL | Request sets the line control register |
| 7 | IOCTL_SERIAL_GET_LINE_CONTROL | To obtain the value of the line control register |
| 8 | IOCTL_SERIAL_SET_CHARS | To set special characters |
| 9 | IOCTL_SERIAL_GET_CHARS | Request retrieves the special characters that the serial controller driver uses with handshake flow control |
| 10 | IOCTL_SERIAL_SET_HANDFLOW | To set Configuration of handshake flow control |
| 11 | IOCTL_SERIAL_GET_HANDFLOW | Request returns information about the configuration of the handshake flow control |
| 12 | IOCTL_SERIAL_GET_MODEMSTATUS | Request updates the modem status and return value of the modem status register before the update |
| 13 | IOCTL_SERIAL_GET_DTRRTS | Request return information about the data terminal ready (DTR) control signal and request to send (RTS) control signal |
| 14 | IOCTL_SERIAL_GET_COMMSTATUS | Request returns information about the communication status of a serial device |
| 15 | IOCTL_SERIAL_GET_PROPERTIES | Request returns information about the capabilities of a serial controller. |
| 16 | IOCTL_SERIAL_SET_FIFO_CONTROL | Request sets the FIFO control register (FCR) |
| 17 | IOCTL_SERIAL_GET_STATS | Request returns information about the performance of a serial controller |

| | | |
|---|---|---|
| 18 | IOCTL_SERIAL_CLEAR_STATS | Request clears the performance statistics for a serial device. |
| 19 | IOCTL_SERIAL_PURGE | Request cancels the specified requests and deletes data from the specified buffers. |
| 20 | IOCTL_SERIAL_SET_TIMEOUTS | Request sets the time-out values that the serial controller driver uses for read and write requests. |
| 21 | IOCTL_UARTTESTTOOL_CLOSE | Disable and close Multi-Com Port |
| 22 | IOCTL_SERIAL_GET_TIMEOUTS | Request returns the time-out values that the serial controller driver uses with read and write requests. |
| 23 | IOCTL_SERIAL_SET_WAIT_MASK | Request configures the serial controller driver to notify a client after the occurrence of any one of a specified set of wait events. |
| 24 | IOCTL_SERIAL_WAIT_ON_MASK | Request is used to wait for the occurrence of any wait event specified by using an IOCTL_SERIAL_SET_WAIT_MASK request. |
| 25 | IOCTL_SERIAL_SET_QUEUE_SIZE | Request sets the size of the internal receive buffer |

## 3.3 Error Handling

Since the IOCTL command is implemented using the Windows API, the return value of the call is dependent on and defined by the OS. On Windows, the return value is a non-zero value. If the error is detected within or outside the driver, an appropriate system defined value is returned by the driver.

## 3.4 Programming Guide

This section explains the basic procedure to use the HSUART driver from a user application mode. All operations are performed through the IOCTLs that are exposed by the HSUART driver.

### 3.4.1 Open Device

The HSUART driver is opened using the Win32 CreateFile API. Since there are more than one HSUART Serial Com Port in APL-I Platform, therefore it require configure each of the handler, so can be able open the correct controller com port that user need.

### 3.4.2 Set IOCTLs Configuration and Data Transfer

In HSUART, it supports full duplex, single duplex and others configuration like baudrate, data size, flow control, priority and stop bit. All configures by using Input/Output controller.

WaitforMultipleObjectsEx function is used to waits until one of all of the specified objects are in the signaled state. Example for full duplex:

```
WaitForMultipleObjectsEx(2, writeThread, TRUE, INFINITE, TRUE);
WaitForMultipleObjectsEx(2, readThread, TRUE, INFINITE, TRUE);
```

Below is the IOCTL code sample:

**IOCTL_SERIAL_GET_BAUD_RATE and IOCTL_SERIAL_SET_BAUD_RATE Code Example**

```
ULONG BaudRate_value = 0;
DWORD junk;

status = DeviceIoControl(
                fileHandler,
                IOCTL_SERIAL_GET_BAUD_RATE,
                NULL,0,
                &BaudRate_value,
                sizeof(BaudRate_value),
                &junk,
                (LPOVERLAPPED)NULL);

status = DeviceIoControl(
                fileHandle,
                IOCTL_SERIAL_SET_BAUD_RATE,
                &BaudRate_set,
                sizeof(BaudRate_set),
                NULL,0,
                &junk,
                (LPOVERLAPPED)NULL);
```

**IOCTL_SERIAL_SET_RTS Code Example**

```
status = DeviceIoControl(
                fileHandler,
                IOCTL_SERIAL_SET_RTS,
                NULL,0,
                NILL,0,
                &junk,
                (LPOVERLAPPED)NULL);
```

### IOCTL_SERIAL_SET_LINE_CONTROL Code Example

```
SERIAL_LINE_CONTROL LineCtl;
LineCtl.Parity = Parity;
LineCtl.StopBits = StopBits;
LineCtl.WordLength = WordLength;

status = DeviceIoControl(
                FileHandle,
                IOCTL_SERIAL_SET_LINE_CONTROL,
                &LineCtl,
                sizeof(SERIAL_LINE_CONTROL),NULL,0,
                &junk,
                (LPOVERLAPPED)NULL);
```

### IOCTL_SERIAL_SET_HANDFLOW Code Example

```
SERIAL_HANDFLOW HandFlow;
HandFlow.ControlHandShake = ControlHandShake;
HandFlow.FlowReplace = FlowReplace;
HandFlow.XoffLimit = XonLimit;HandFlow.XonLimit = XoffLimit;

status = DeviceIoControl(
                Filehandle,
                IOCTL_SERIAL_SET_HANDFLOW,
                &HandFlow,
                sizeof(SERIAL_HANDFLOW),NULL,0,
                &junk,
                (LPOVERLAPPED)NULL);
```

### IOCTL_SERIAL_SET_MODEM_CONTROL Code Example

```
status = DeviceIoControl(
                FileHandle,
                IOCTL_SERIAL_SET_MODEM_CONTROL,
                &ModermCode,
                sizeof(ModermCode),NULL,0,
                &junk,
                (LPOVERLAPPED)NULL);
```

### IOCTL_SERIAL_SET_CHARS Code Example

```
SERIAL_CHARS SpecialChars;
SpecialChars.BreakChar = BreakChar;SpecialChars.EofChar = EofChar;
SpecialChars.ErrorChar = ErrorChar;SpecialChars.EventChar = EventChar;
SpecialChars.XoffChar = XoffChar;SpecialChars.XonChar = XonChar;

status = DeviceIoControl(
                FileHandle,
                IOCTL_SERIAL_SET_CHARS,
                &SpecialChars,
                sizeof(SERIAL_CHARS),NULL,0,
                &junk,
                (LPOVERLAPPED)NULL);
```

### IOCTL_SERIAL_SET_FIFO_CONTROL Code Example

```
ULONG FifoControl = 0x86;

status = DeviceIoControl(
                FileHandle,
                IOCTL_SERIAL_SET_FIFO_CONTROL,
                &FifoControl,
                sizeof(FifoControl),NULL,0,
                &junk,
                (LPOVERLAPPED)NULL);
```

### Example of write and read transfer:

```
    writeThread[0] = CreateThread(
                    NULL,              // default security attributes
                    0,                 // use default stack size
                    ThreadWriteCom,    // thread function name
                    hHandle,           // argument to thread function
                    0,                 // use default creation flags
                    &dwThreadIdW1);    // returns the thread identifier

    readThread[0] = CreateThread(
                    NULL,
                    0,
                    ThreadReadCom,
                    hHandle,
                    0,
                    &dwThreadIdR1);
```

### 3.4.3 Close Device

Once all operations related to the HSUART driver are finished the device handle must free the application by calling the API CloseHandle.

```
CloseHandle(hHandle);
```

§