AmpedUp Target Manager Reference Guide

28 April 2011

Amp'edUP Target Manager Reference Guide

TABLE OF CONTNTS

1 Preface 5

1.1 Purpose 5

1.2 Audience 5

1.3 Reference 5

2 OVERVIEW 7

2.1 Overview of Target Manager Features 7

3 GPIO FUNCTIONS 8

3.1 void Targ_GPIO_ISR(uint32 ExtLineSource) 8

3.2 void Targ_InitPio(void) 8

3.3 Config 8

3.4 GPIO Configure, Read, and Write Functions 8

3.5 General Purpose GPIO Write and Read Functions 8

4 UART 9

4.1 Targ_UartDriverInitialization 9
4.1.1 Driver parameters for Main UART 9

4.2 Targ_SetupUart 9
4.2.1 User Parameters 9

4.3 Targ_UartConfigure 9

5 LOW POWER MODE 10

5.1 Targ_Idle 10

5.2 Targ_GotoSleepMode 10

5.3 Targ_CpuSleep 10

6 MEMORY 11

6.1 Flash Memory Map 11
6.1.1 Custom NVM Parameters 11

6.1.2 Defining Record IDs 11
6.1.3 Writing to Records 11
6.1.4 Reading from Records 12

6.2 RAM Memory 12
6.2.1 Showpool 12
6.2.2 Showmem 12

7 OS TIMERS 13

7.1 OS_TimerRegisterRequest (uint32 Transac, uint16 ReqId, uint32 MsTmr, tBP_SignalNo SigNo) 13

7.2 OS_CancelTmrRequest(uint16 ReqId) 14

1 Preface

1.1 Purpose

This publication, *AmpedUp Target Manager Reference Guide*, serves as a reference for AmpedUp SDK developers desiring to integrate the various hardware target platform features into an application. The description and usage of each feature is detailed in the document.

1.2 Audience

This document is written for the embedded applications developer using the AmpedUp software to interface to Amp'ed RF's abSerial platform. The programmer should be familiar with Bluetooth profiles, driver software, and various microcontroller concepts.

1.3 Reference

The following definitions are referred to in this release:

Table 1. Definitions and Acronyms

Term	Description/Meaning
API	Application Programmer Interface
AT	Text based command standard commonly used for modems
BD	Bluetooth Device
BdAddr	Bluetooth Device Address
DUN	Dial-up Network Profile
GAP	Generic Access Profile
HCI	Host Controller Interface
ID	Identification
Inq.	Inquiry
IPCP	Internet Protocol Control Protocol
L2CAP	Logical Link Control and Adaptation Protocol
LS	Least Significant
OS	Operating System
PIN	Personal Identification Number
RAM	Random Access Memory
Req	Request
RFCOMM	Serial Port Emulation
RTOS	Real Time Operating System
SDAP	Service Discovery Application Profile
SDP	Service Discovery Protocol
secs	Seconds
SPP	Serial Port Profile
TCP	Transport Control Protocol
UART	Universal Asynchronous Receiver-Transmitter

Term	Description/Meaning
ULS	Upper Layer Stack
USB	Universal Serial Bus
UUID	Universally Unique Identifier

2 Overview

2.1 Overview of Target Manager Features

The Target Manager includes support for the following features:

General purpose I/O pins
Serial interfaces: UART, SPI, I2C, I2S
Flash and RAM memory
External interrupts
Low power operating modes
Timers

These features are supported using a STM32 microcontroller from ST Microelectronics. Further details about this controller may be found at: http://www.st.com/mcu/inchtml-pages-stm32.html

3 GPIO Functions

3.1 void Targ_GPIO_ISR(uint32 ExtLineSource)

This function handles the GPIO interrupt exception and generated a notification OS signal to the application layer for this event.

The following 9 GPIO pins may be used as interrupts, based on the BT11 or BT31 modules:

GPIO: 0, 1, 2, 3, 4, 6, 10, 12, 14

Other GPIO lines may be used as inputs, outputs, our alternative functions, but not as an interrupt.

3.2 void Targ_InitPio(void)

3.3 Config

This function sets up the GPIO functionality. Parameter details can be found in the STM32 GPIO documentation.

3.4 GPIO Configure, Read, and Write Functions

TM_SetPio(uint16 GPIONumber)

TM_ResetPio(uint16 GPIONumber)

TM_SetPioAsOutptut(uint16 GPIONumber)

TM_SetPioAsInput(uint16 GPIONumber)

These functions allow the SDK application to utilized the module's GPIO naming scheme to control the various GPIO pins. The STM32 port identification scheme is not used with these tools.

3.5 General Purpose GPIO Write and Read Functions

These functions utilize the STM32 port naming scheme to read and write to any pin in the STM32 controller. These should be used with caution, and the above SDK naming scheme is generally safer to use.

GPIO_BitRead(GPIO_TypeDef* GPIOx, uint16 pin)

GPIO_WriteBit(GPIO_TypeDef* GPIOx, uint16 pin)

Parameters

GPIOx: This is the GPIO port of the STM32.

Pin: This is the pin designator of the port. It will be converted to a bit field.

Val: Value to write to the pin, 1 or 0.

4 UART

4.1 Targ_UartDriverInitialization

The purpose of this function is to initialize the UART driver values. These values are based on the UartType argument passed in the function call.

4.1.1 Driver parameters for Main UART

UartPin.Rx, UartPin.Tx, UartPin.RTS, UartPin.CTS: These are the I/O pin assignments. They must be value options based on one of the STM32 UARTs.

UartTx.DmaChannel, UartTx.NVIC_IRQChannel, UartRx.DmaChannel, UartRx.NVIC_IRQChannel, UartTx.ITPending: These are the DMA mapping and interrupt channels.

UartRx.Timeout: This is the number of bit lengths an idle Rx Uart will wait before generating an interrupt.

UartRx.FlowCntrlEnable, UartTx.FlowCntrlEnable: Enable the RTS and CTS flow control features.

UartRx.DmaBuffer: Points to the DMA memory buffer.

UartRx.DmaSize: The number of DMA bytes.

UartRx.DmaLowWM: This is the lower threshold to assert the RTS pin.

UartRx.PoolBufNum: The number of Rx buffers available.

UartRx.credit_rx: This is the RFCOMM layer's credit based flow control maximum value. It should be equal or less than the number of available buffers.

UartRx.PoolBufSize: Number of bytes in each Rx Buffer. This should contain an extra 44 bytes for lower layer and header overhead data.

UartRx.pPool: Not used.

UartRx.PoolReleaseFunction: Internal notification function – do not change this.

UartRx.SWI, UartRx.SWIChannel: The UART interrupt channel mapping.

4.2 Targ_SetupUart

This function sets up the user parameter values of the UART. The UART type is passed in the UartType field.

4.2.1 User Parameters

UartDataBits: number of data bits in a character, 5 to 8.

UartStopBits: width of the stop bit, 1 or 2 bits.

UartParity: none, even, or odd parity.

BaudRate: setting from 1200 bps to 2000000 bps (2M bps).

4.3 Targ_UartConfigure

This function is the call to enable the UART and begin it's usage. The UartType and initial user parameters are passed in.

5 Low Power Mode

5.1 Targ_Idle

This function is called when the OS had nothing further to process or schedule. A 22ms Deep Sleep Hold-off timer is used to set a minimum active time.

5.2 Targ_GotoSleepMode

This function begins the Deep Sleep process by sending a break character on the UART Tx line to the Bluetooth controller.

The Bluetooth controller signals when it's ready, and then the CPU is switched into Deep Sleep mode as well.

5.3 Targ_CpuSleep

This function places the CPU into Deep Sleep mode. The Bluetooth controller is handled separately. See the STM32 low power mode documentation for further details.

6 Memory

The STM32 controller contains a total of 48K RAM and 256K bytes of Flash memory. The Amp'edUP stack normally utilizes, 43K RAM and 196K Flash, which allows the remaining amounts to be use for the SDK custom application purposes:

5K RAM and 60K Flash

Certain features, such as Mulipoint and Apple IOS support, will result in less available user memory. Memory usage may also vary across releases, but the above minimum levels are "guaranteed" available values.

6.1 Flash Memory Map

The Flash memory is organized according to the follow memory map:

Table 1: Flash Memory Map

8000000-8002000 (8K bytes)	Boot Loader and BD Address
8002000-803E000 (240K bytes)	Application
803E000-8040000 (8K bytes)	User configuration parameters

6.1.1 Custom NVM Parameters

A user may read and write to NVM records using our custom record file system. Each record may contain from 1 to 460 bytes. The total amount of space available is 5K remaining (from the 8K User configuration above).

6.1.2 Defining Record IDs

Each Custom Record for NVM must be defined before usage. The initial ID base is defined as:

```
#define TMFM_1ST_CUSTOM_ID  0x3001  // Custom record ID base
```

Each record must have its own pre-defined ID.

6.1.3 Writing to Records

The following functions are use to write to a Record:

void * TM_GetFlashWritePtr(uint16 Id, uint16 Size)

void TM_FlashWriteDone(uint16 Id)

Step 1: Prepare a memory structure to hold the record's data
Step 2: Call the TM_GetFlashWritePtr function and get the pointer to the custom record memory.
Step 3: Load your custom data structure into the memory pointed to by the result from Step 2.
Step 4: Call the TM_FlashWriteDone function to save your data.

6.1.4 Reading from Records

The following functions are use to read a Record's data:

void * TM_GetFlashReadPtr(uint16 Id)

Step 1: Prepare a memory structure to hold the record's data
Step 2: Call the TM_GetFlashReadPtr function and get the pointer to the custom record memory.

6.2 RAM Memory

The RAM memory is allocated into two main groups: global static memory and memory pools for the OS/User. The following commands may be used to view memory allocations:

6.2.1 Showpool

This command displays the various pool memory allocations.

```
Pool          Alloc     Peak    Num   Size   Total
MainRxPool        1        1      6    376    2256
HciRxPool         0        1      5    396    1980
GkiMem3           0        2      4    660    2640
GkiMem2           0        1      2    384     768
GkiMem1           1        1      4    256    1024
GkiMem0           0        6     12     64     768
TMList            1        4     30     28     840

MEMORY
  Useable: 10276
 Overhead:  1008
    Total: 11284
```

Note: these allocations may change depending on the FW version.

6.2.2 Showmem

This command displays the total memory allocation.

at+ab showmem

Stack: 0x20000130-0x20000900 - 1168 free 832 used 2000 total

Memory: 0x200060A0-0x2000FE00 - 23284 free 17004 used 40288 total

SRAM Use: 0x20000000-0x2000a30c - 40.7 K

Allocation: 11284 pool 5720 reserved

Stack: memory used by the system OS.

Memory: *Free* memory is RAM that is not reserved for static or pool memory, *Used* is the *pool* + *reserved* memory below, and *total* is the *free* + *used*.

Allocation: *Pool* memory is the same value from ShowPool above, *reserved* is memory used for the Amp'edUP stack control blocks.

Note: these allocations may change depending on the FW version, but only 5K free RAM bytes are "guaranteed".

7 OS Timers

The RTOS of Target Manager supports a general OS timer function. The minimum tick granularity is 1ms, with potential 2ms latency. For high accuracy or timers < 10ms, it's recommended to use a dedicated STM32 timer, and not the RTOS general timer.

7.1 OS_TimerRegisterRequest (uint32 Transac, uint16 ReqId, uint32 MsTmr, tBP_SignalNo SigNo)

Parameters

Transac: This is a unique "seed" for each timer. Generally, this can be left as 0. For frequently repeated timers, it may be better to use a counter however. Multiple simultaneous timer of the same ID may be resolved further by this field.

ReqId: This is the timer ID value, defined in the timer enumeration from the App_Main.h

(example)

```
//-------------------------------------
//   Defines and enums
//-------------------------------------
#define TM_TMRID_APP_BASE   0xC000
#define SIG_APP_BASE        60000

enum {
    APP_TMRID_HELLO = TM_TMRID_APP_BASE,
    APP_TMRID_SEND_HCI_MESSAGE,
    APP_TMRID_DEBOUNCE,
    APP_TMRID_BOND,
    APP_TMRID_DISCOVERY,
    APP_TMRID_CONNECT_INTERVAL,
    APP_TMRID_STARTED,
};
```

MsTmr: Value of this timer in millisecond ticks

SigNo: This is the signal to be send when this timer expires, defined in the signal enumeration from the App_Main.h

(example)

```
enum {
    SIG_APP_HELLO = SIG_APP_BASE,
    SIG_SEND_HCI_MESSAGE,
    SIG_APP_GPIO_INTERRUPT,
    SIG_APP_GPIO_DEBOUNCE,
    SIG_APP_ANOTHER,
    SIG_APP_MESSAGE,
    SIG_APP_BOND,
    SIG_APP_DISCOVERY_TIMEOUT,
    SIG_APP_CONNECT_INTERVAL_TIMEOUT,
    SIG_APP_STARTED,
```

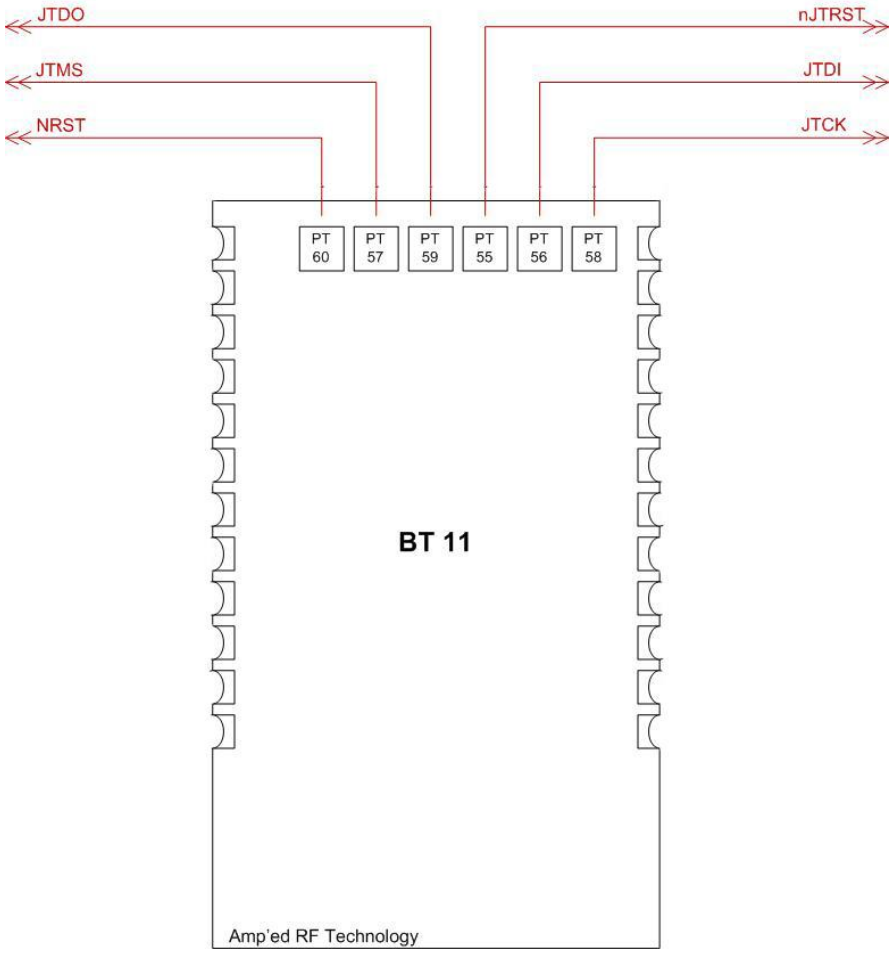
```
};
```

7.2 OS_CancelTmrRequest(uint16 ReqId)

This function will cancel a timer, using the given timer ID in `ReqId`. It should be cautioned that it's possible to cause a race condition, because the timer may have already expired but without the resultant signal arriving yet.

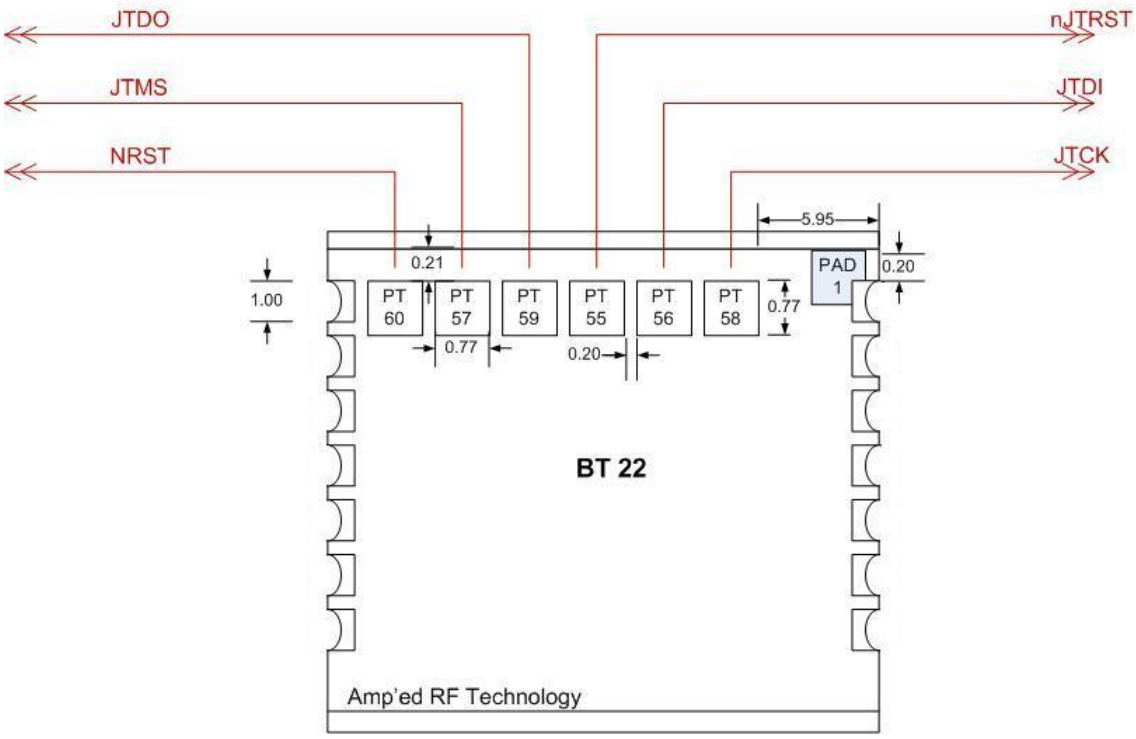
JTAG Interface

JTAG Interface Pads



Bottom View of BT 11

JTAG Interface Pads



Bottom View of BT 22