The Ag102 STAT output can be used for visual indication (with an external LED) or by a μ-controller for automatic status monitoring, see Figure 1.

![Diagram of Ag102 STAT output with LED driver and μ-controller](image)

**Figure 1: Ag102 STAT output**

The LED drive current is limited by an internal 1K resistor (and the +5V internal reference voltage), but this can easily be increased with an external transistor, if required.

**Mode 0 – Normal Operation**

During a normal charge cycle the ‘STAT’ output will be a steady state ‘Logic 1’ (Mode 0).

After the Ag102 has entered the float cycle it will periodically check the battery capacity - approximately once an hour. It does this by stopping the top-up pulses for a short duration and checking the battery terminal voltage response.

If the battery terminal voltage does not drop below the capacity threshold, the Ag102 will resume operation in Mode 0 and the STAT output will remain at a steady state ‘Logic 1’.

**Mode 1 – Battery Capacity Warning**

If the terminal voltage drops below the threshold level (during the battery capacity check), then the Ag102 will go into Mode 1.

In Mode 1 the STAT output will change to indicate a battery capacity low warning by generating an inverse pulse (‘Logic 0’) for ~100ms (then returns to ‘Logic 1’). The interval between these (inverse) pulses can vary, but will usually be < 10 seconds.
It is important to remember that even if the Ag102 detects a low battery capacity it will continue to charge the battery. Mode 1 is a warning that the battery capacity is getting low and that the battery may need to be changed. If the Ag102 detects an error condition (Modes 2 to 4), then the STAT output will go to ‘Logic 0’ for 1 second then will send ‘Logic 1’ pulse(s) with a ~200ms mark (and a ~200ms space between pulses), which will be repeated with ~1 second gap (Mode 3 does have an exception to this which is described in Section 5.5.4).

Mode 2 – Defective or Disconnected battery

If the battery is disconnected or is completely defective, the Ag102 will go into Mode 2 and cycle here until a healthy battery is connected. When a healthy battery is reconnected the Ag102 will return to Mode 0 (normal operation) unless the Ag102 detects a problem.

Mode 3 – Over Temperature

If a battery over temperature condition occurs, the Ag102 will shutdown its DC-DC converter to protect the battery and will go into Mode 3. The STAT pin will output five sets of two pulses with the standard 1s delay in between each set of pulses. But after the fifth set of pulses, the Ag102 will restart to check the temperature during an extended ‘Logic 0’ period (> 3seconds). If the battery is still over temperature the Ag102 will shut down and continue to cycle on Mode 3. When the Ag102 detects that the battery temperature has dropped below 50°C (the maximum operating temperature), the part will return to Mode 0 (normal operation).

Mode 4 – Over Current

If an output over current condition is detected, the Ag102 will again shutdown its DC-DC converter and will go into Mode 4. This is considered to be a major fault condition and the Ag102 will need to be power cycled to resume normal operation to protect the battery and itself.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Status Mode</th>
<th>STAT Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Normal Operation</td>
<td>Steady State ‘Logic 1’</td>
</tr>
<tr>
<td>1</td>
<td>Battery Capacity Warning</td>
<td>1 Inverse Pulse</td>
</tr>
<tr>
<td>2</td>
<td>Defective or Disconnected Battery Error</td>
<td>1 Pulse</td>
</tr>
<tr>
<td>3</td>
<td>Over Temperature Error</td>
<td>2 Pulses</td>
</tr>
<tr>
<td>4</td>
<td>Over Current Error</td>
<td>3 Pulses</td>
</tr>
</tbody>
</table>

Table 1: Ag102 STAT output codes
When using a μ-controller the STAT output needs to be monitored and any error codes handled when detected. This can be done easily as demonstrated by the example shown in Figure 3 which uses a very basic PIC10F200 - 8-Bit flash μ-controller from Microchip.

The “Status” input shown in Figure 3 connects directly to the “Status” output shown in Figure 1-b. Table 2 shows the output status of the GP0 – GP2 pins when using the example code shown below.
### Table 2: GP0 – GP2 Outputs

<table>
<thead>
<tr>
<th>STAT Output</th>
<th>Mode</th>
<th>GP2</th>
<th>GP1</th>
<th>GP0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Inverse Pulse</td>
<td>Battery Low Warning</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1 Pulse</td>
<td>Battery Disconnected</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2 Pulses</td>
<td>Over Temperature</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3 Pulses</td>
<td>Over Current</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Steady State ‘Logic 1’</td>
<td>Normal Operation</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The code shown below is purely for demonstrative purposes, but can be adapted to suit your own µ-controller and application.

```asm
;**********************************************************************
; Filename:     Battery-Status-4.asm                                *
; Date:               09/04/09                                            *
; File Version:                                                      *
; Author:             Tony Morgan                                         *
; Company:            Silver Telecom Ltd                                *
;**********************************************************************
;    Files Required: P10F200.INC                                        *
;**********************************************************************

list      p=10F200            ; list directive to define processor
#include <p10F200.inc>        ; processor specific variable definitions
__CONFIG   _MCLRE_OFF & _CP_OFF & _WDT_OFF & _IntRC_OSC
__idlocs 0x0100

; '__CONFIG' directive is used to embed configuration word within .asm file.
; The labels following the directive are located in the respective .inc file.
; See respective data sheet for additional information on configuration word.

RAMTRIS    RES   1 ; this is a baseline part so have to create
             ; own tris register in RAM to keep track of
             ; input and output pins (very important!)

;**********************************************************************
;***** VARIABLE DEFINITIONS
;**********************************************************************

NormalMode equ 0x00
WarningMode equ 0x01
ErrorMode equ 0x02
STATHigh equ 0x03
STATLow equ 0x04
ModeChanged equ 0x07

cblock 10h ; list of variables used in the program

Delay1     ; three delay loop bytes
Delay2
Delay3
Counter    ; loop counter
LowCount   ; low loop counter
Pulses     ; number of pulses
StatusFlag ; status flag
WarningCounter ; warning message 10 seconds counter

ende

;**********************************************************************
; ORG     0x0000 ; coding begins here
;**********************************************************************

start

movlw b'00000000'
```
Ag102 STAT Output Decoder

movwf OSCCAL ; update register with factory cal value

movlw b'11011111' ; set options register to allow GP2 to be set as an output
table

movlw b'00001000' ; GP0 - GP2 = Output, GP3 = Input
movwf RAMTRIS

tris GPIO

clrf Pulses ; clear the pulse counter
clrf WarningCounter ; clear the warning counter
clrf StatusFlag ; clear the operating mode flag

startloop
call delay10ms ; delay 10ms
btfss GPIO, 3 ; check if the status pin has gone high
goto startloop ; repeat the startloop until the Ag102 output pin goes high

movlw B'10001001' ; set Normal Mode, STATHigh and ModeChanged
movwf StatusFlag ; update the status byte

mainloop
btfsc StatusFlag, ModeChanged ; check if the mode has changed
call statusupdate ; update the output pins if the mode has changed

movlw d'200' ; set the counter to 200 (x 10ms delay) = 2000ms
movwf Counter

highloop ; loop until STAT goes low or counter = 0
call delay10ms ; delay 10ms
btfss GPIO, 3 ; test the STAT output pin
goto highloop ; repeat the highloop until the counter reaches zero
btfss StatusFlag, STATHigh ; check if the STAT high flag is set
incf Pulses, f ; if it isn't than increment the pulse counter
btfsc StatusFlag, STATHigh ; set the STAT high flag
goto highloop ; repeat highloop until the counter reaches zero
btfss StatusFlag, WarningMode ; check if in warning mode
goto checknormalmode ; jump to check normal mode if not
decfsz WarningCounter, f ; this counter allow STAT to be high for 10s before clearing
goto mainloop ; go back to the main loop if the counter in not zero

checknormalmode
btfsc StatusFlag, NormalMode ; check if already in normal mode, skip the next instruction if not
goto mainloop ; go back to the main loop

movlw B'10001001' ; set Normal Mode, STATHigh and ModeChanged
movwf StatusFlag ; update the status byte
goto mainloop ; go back to the main loop

STATLow ; the STAT output is low

movlw d'015';
movwf Counter ; set the counter to 15 (x 10ms delay) = 150ms

lowloop1
call delay10ms ; delay 10ms
btfsc GPIO, 3 ; test the STAT output pin
goto lowloop1 ; repeat the lowloop1 until the counter reaches zero

movlw d'010'
Ag102 STAT Output Decoder

movwf Counter ; set the counter to 10 (x 10ms delay) = 100ms

lowloop2
call delay10ms ; delay 10ms
btfsc GPIO, 3 ; test the STAT output pin
goto lowloopexit ; exit the loop when this pin goes high
decfsz Counter, f ; decrement the counter
goto lowloop2 ; repeat the lowloop2 until the counter reaches zero
movlw d'250'
movwf Counter ; set the counter to 250 (x 20ms delay) = 5000ms

lowloop3
call delay20ms ; delay 20ms
btfsc GPIO, 3 ; test the STAT output pin
goto highSTAT1 ; exit the loop when this pin goes high
decfsz Counter, f ; decrement the counter
goto lowloop3 ; repeat the lowloop3 until the counter reaches zero
movlw B'10010000' ; set STATLow flag and the Mode Changed flag
movwf StatusFlag ; update the status byte
goto mainloop ; return to the mainloop

LowSTAT1
btfss StatusFlag,ErrorMode ; check if already in error mode, skip the next instruction if not
goto LowSTAT2
movlw B'10000100' ; set ErrorMode and the Mode Changed flag, clear the STATHigh flag
movwf StatusFlag ; update the status byte
goto mainloop ; return to the mainloop

LowSTAT2
movlw B'00000100' ; set ErrorMode, clear the Mode Changed flag and STATHigh flag
movwf StatusFlag ; update the status byte
goto mainloop ; return to the mainloop

Lowloopexit
movlw B'00000100' ; set ErrorMode, clear the STATHigh flag and clear the Mode Changed flag
movwf StatusFlag ; update the status byte
goto mainloop ; go back to the main loop with STAT set low

Warningpulse
movlw B'10001010' ; set WarningMode, STATHigh and ModeChanged
movwf StatusFlag ; save in StatusFlag
movlw D'005' ; set the warning counter to 5 x 2s = 10s
movlw B'00000001' ; set Pulses to 1 to indicate single warning pulse
movwf Pulses ; save in Pulses

;**********************************************************************
; Update outputs pins relative to the lower three Pulses register bits
;**********************************************************************

statusupdate
btfss StatusFlag,ModeChanged ; Test if the mode has changed
goto statusupdateexit ; jump to exit if the mode is unchanged
btfsc StatusFlag,STATLow ; Test if the warning mode has been set
goto InvalidMode ; goto invalid mode check output steady low
btfss StatusFlag,WarningMode ; Test if the warning mode has been set
goto errorcheck ; goto error mode check if it hasn't
movlw b'0000001' ; set Pulses to 1 to indicate single warning pulse
movwf Pulses ; save in Pulses

Page 6 of 9
AN102-3v1-2
goto outputchanged ; jump to set the output

errorcheck

btfss StatusFlag,ErrorMode ; Test if the error mode has been set
goto normalcheck ; goto normal mode check if it hasn't

incf Pulses, f
goto outputchanged ; jump to set the output

normalcheck

btfss StatusFlag,NormalMode ; Test if the normal mode has been set
goto InvalidMode ; goto invalid mode if non of the mode flags have been set

movlw b'00000111' ; set all the outputs high to indicate normal mode
movwf Pulses ; save in Pulses
goto outputchanged ; jump to set the output

InvalidMode

movlw b'00000000' ; clear all the outputs to indicate an invalid mode
movwf Pulses ; save in Pulses

outputchanged

btfsc Pulses,0 ; test if the bit has been cleared
bsf GPIO,0 ; clear output GP0 if true
btfss Pulses,0 ; test if the bit has been set
bcf GPIO,0 ; set GP0 if true

btfsc Pulses,1 ; test if the bit has been cleared
bsf GPIO,1 ; clear output GP1 if true
btfss Pulses,1 ; test if the bit has been set
bcf GPIO,1 ; set GP1 if true

btfsc Pulses,2 ; test if the bit has been cleared
bsf GPIO,2 ; clear output GP2 if true
btfss Pulses,2 ; test if the bit has been set
bcf GPIO,2 ; set GP2 if true

clrwf Pulses ; clear the Pulses byte before returning
bcf StatusFlag,ModeChanged ; clear the mode changed flag

statusupdateexit

retlw 0

**********************************************************************
* delay routine using simple loops
**********************************************************************
delay100ms

movlw D'100'
movwf Delay3

goto delayLoop3

delay20ms

movlw D'020'
movwf Delay3

goto delayLoop3

delay10ms

movlw D'010'
movwf Delay3

goto delayLoop3

delay1ms

movlw D'001'

movwf Delay3

delayLoop3
    movlw 0x02
    movwf Delay2

delayLoop2
    movlw 0x95
    movwf Delay1

delayLoop1
    decfsz Delay1, f
    goto delayLoop1

decfsz Delay2, f
    goto delayLoop2

decfsz Delay3, f
    goto delayLoop3
retlw 0

END ; directive 'end of program'
The PIC10F200 shown in Figure 3 (with the code detailed above) can be connected to a 3 to 8 decoder as shown in Figure 4. Alternatively the 74HC138 outputs can be connected to the main system μ-controller.

![Diagram](image)

**Figure 4: μ-controller with decoder**

The way in which outputs GP0 – GP2 respond to the error code can be modified by changing the code in the “statusupdate” section.

A copy of the ‘Bat-Status-4.asm’ file is available upon request.