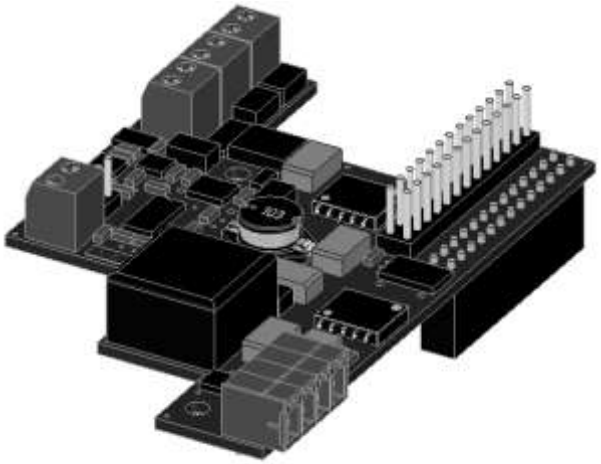
# Battery shield for Raspberry Pi and DV0 expansion boards controller

## Features

- Power supply for Raspberry Pi A and B models
- Wide input voltage for battery or wall adapter
- Safe shutdown for Linux operating system
- Battery voltage monitoring
- Real time clock keeping
- Programmable wake up scheduling
- Programmable hardware restart
- Programmable shutdown and boot time
- Watch dog reset function
- On/Off Push button for manual or remote power up/down sequences
- DV0 bus connector for expansion boards
- Free available Linux commands and API library (Python, C++ or Java) for setting parameters and accessing expansion boards
- Shape designed for easy access to Raspberry Pi GPIO, DSI and CSI Camera connectors
- Spacers and screws for mechanical assembly are included



## Technical specifications

- Power to Raspberry Pi: 5V, 0.8A
- Power to DV0 expansion bus: 5V, 1A
- Both 5V outputs are short circuit protected
- Input voltage for battery or wall adapter: from 6.5V up to 27V
- Both wall adapter and battery connector protected against wrong polarity connection
- Battery chemistry: NiMH or Lead-Acid
- Battery slow charge internal resistor
- Connector for increasing battery charge current
- Power consumption during sleep (all 5V power supplies off): 3.5mA
- Programmable voltage failure threshold
- Selectable mode wall or mode battery standalone
- Real time clock accuracy:0.01%
- Up to 16 expansion boards management capability
- Command latency from user application to IO expansion boards: 15ms

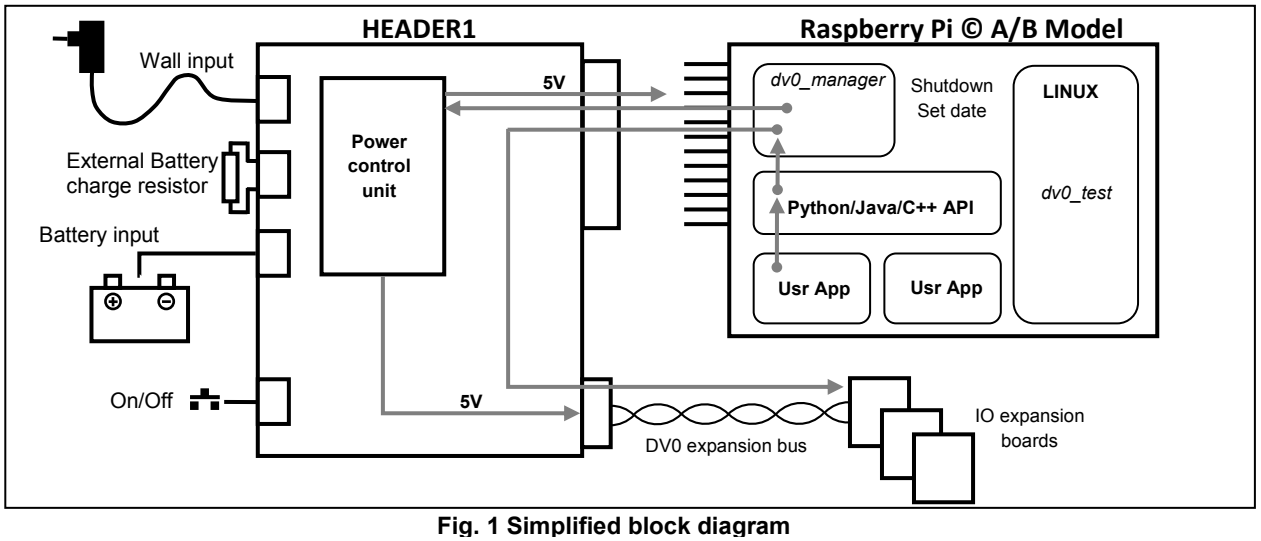



**Fig. 1 Simplified block diagram**

## Table of contents

Revision history

V1.0 Jan-2015

# 1. FUNCTIONAL OVERWIEW

The DeValirian Header1 is a dual source power supply for Raspberry Pi providing two functions:

- Uninterruptible power supply with controlled Linux shutdown and programmable restart features
- Managing DVO Input/Output expansion boards

## 1.1 Uninterruptible power supply.

As an uninterruptible power supply, the Header1 generates stabilized 5V to the Raspberry Pi from the Wall input or the Battery input, whichever is greater. The Header1 control unit communicates to a program running in the Raspberry Pi Linux (*dv0_manager*) which is responsible for issuing a shutdown command when the Wall input voltage falls below the Battery input voltage or when the Battery input voltage falls below a programmable threshold. Thus, file system corruption due to sudden power outage can be avoided.

The Header1 control unit communicates with the free *dv0_manager* program running in the Raspberry Pi Linux using the serial channel ttyAMA0. This channel can be found in the Raspberry expansion connector. See Section 4.0 for more information about *dv0_manager* installation and features.

### 1.1.1 Wall Mode vs. Battery Mode

The Header1 can react in two ways when the power in the Wall input disappears, depending on the state of the jumper JP1. See Fig. 2 to locate this jumper.

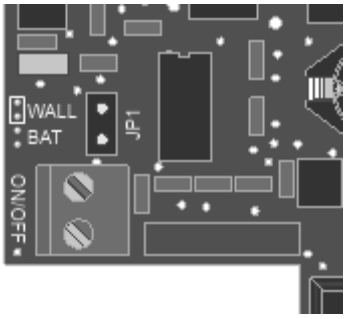



**Fig. 2 JP1 Wall/Battery mode jumper selector**

When this jumper is closed (Wall Mode), the Header1 control unit issues a shutdown command to the Raspberry Pi board whenever the voltage on the Wall Input is lower than the voltage in the Battery Input for more than five seconds. In this mode, the battery size is dimensioned for powering the Raspberry Pi just during the shutdown time (20 to 30 seconds).

The Wall Mode is intended for applications where the main power source comes from a wall adapter supply and a battery is needed only for safe shutdown purposes or sleeping features as time keeping, scheduled restart, on/off pushbutton, etc.

When JP1 is left open (Battery Mode), the shutdown command is issued only when the Battery input voltage falls below *min_voltage* for more than five seconds, no matters what voltage is at Wall input. The default voltage level for *min_voltage* is 6V, but other values can be set programmatically to accommodate a wide range of battery voltages.

The Battery Mode is intended for applications where the main power source is a battery, and Wall Input is used just for slow battery charging or for feeding the board when the battery needs to be temporally disconnected.

The 5V power to Raspberry Pi is always obtained from the input source that exhibits the upper voltage value and automatic balance is made when one source falls below the other one without affecting the stability of Raspberry Pi power supply.

Every time the Header1 control unit sends a shutdown command, it starts a timer and when it reaches *shutdown_time*, the 5V power to Raspberry Pi and DV0 Bus© as well are disconnected. During this time, no power decisions are taken. Also, when start conditions are reached, another timer is launched and no power decisions are taken until this timer reaches *boot_time*. This ensures that safe halt and safe boot process can be done by the Linux, independently of power fluctuations. Both *shutdown_time* and *boot_time* parameters can be set programmatically, as described in Section 4.

### 1.1.2 Power Off Conditions

Conditions that trigger a Power Off Process when the Header1 is at the On State are:

- V(Wall input) < 6 for more than 5 seconds when Wall Mode is selected
- V(Battery input) < *min_voltage* for more than 5 seconds when Battery Mode is selected
- Push button is pressed for more than 4 seconds (unless that *-no_push_active* parameter is set)
- A SIGPWR is sent to dv0_manager
- The *PowerOff()* API function is called by an application connected to *dv0_manager*
- Watch dog reset timeout: when parameter *-watch_dog <watch_dog_time>* has been set and no watch dog reset has been issued for *watch_dog_time* second (see Section 4 for more information)

### 1.1.3 Power Off Process

The Power Off Process starts when one of the Power Off Conditions becomes true. The sequence of steps of this process is:

- Issue a shutdown command to *dv0_manager*, witch calls the Linux command "*shutdown –h now*"
- Cut off the voltage in the DV0 Bus if parameter *-cut_vbus* is set
- Wait for *shutdown_time* seconds
- Cut off the voltage of Raspberry Pi and DV0 Bus
- Go to Sleep State

### 1.1.4 Power On Conditions

Conditions that trigger a Power On Process when the Header1 is at the Sleep State are:

- V(Wall input) > 6.2V for more than 5 seconds when Wall Mode is selected
- V(Battery input) > *min_voltage+0.2* for more than 5 seconds when Battery Mode is selected
- Push button is pressed for more than 2 seconds (unless that *-no_push_active* parameter is set)
- Restart timer timeout: when *-restart_time <time_to_restart>* has ben set and *time_to_restart* minutes have been passed until the last arriving to Sleep State
- Scheduled start: parameter *star_up_time* matches the current date and time kept by Header1 (see Section 4)

### 1.1.5 Power On Process

The Power Off Process starts when one of the Power On Conditions becomes true. The sequence of steps of this process is:

- Rise Raspeberry Pi voltage and DV0 bus voltage to 5V
- Wait boot_time seconds
- Send the kept date and time to *dv0_manager* if parameter *-force_date* has been set, which sets system time
- Executes the Discovery Process
- Go to On State

### 1.1.6 On State features

When the Header1 is in the On State, it accepts the following commands:

- Set current date and time
- Set restart time (*restart_time*)
- Set scheduled start (*start_up_time*)
- Reset watch dog timer if enabled
- Route commands to/from DVO bus expansion boards

The Figure 3 shows the Header1 states and their transitions.

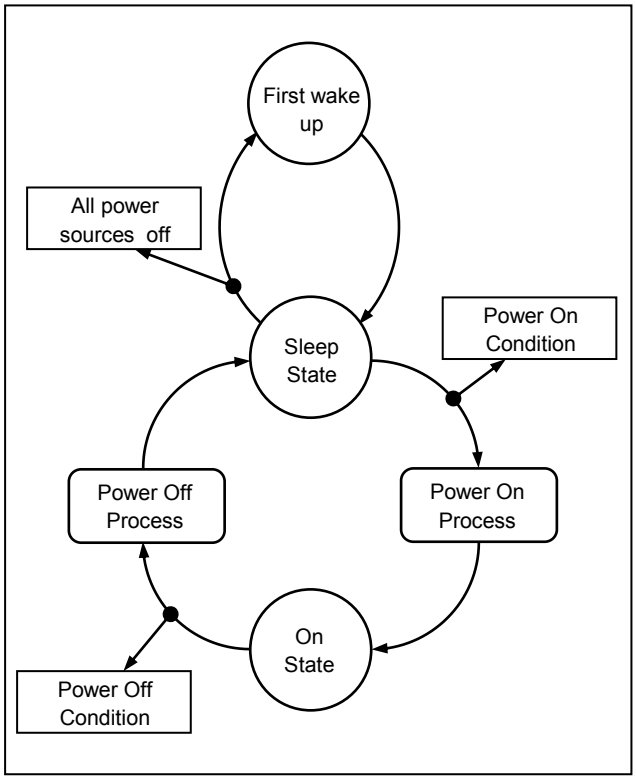



**Fig. 3 Header1 states and transitions**

### 1.1.7 Real time clock

The Header1 board keeps the date and time during both On and Sleep states. It starts at 1/1/2001 00:00:00 on the First Wake Up and can be set to any value by using the free program *dv0_test* or by calling the API function *SetDate()*

The accuracy of the Header1 real time clock is 0.01%, which is not a good long term accuracy because it means that the time error over 24 hours can be up to 1 minute and 20 seconds (it depends on ambient temperature).

However, this accuracy is good enough to starts the board at scheduled time and to give an initial date to the Raspberry Pi. With this initial date, the Linux OS can take decisions and, with relaxed periodicity, get access to Internet looking for a NTP server and to synchronize itself and the Header1 board.

## 1.2 Managing DVO IO expansion boards

DVO IO Expansion boards allow Linux based systems to gain access of the real word by giving easy control of physical inputs and outputs, like analog or digital sensors, relays, stepper motors, servo motors, keyboard, displays, etc

These boards need a 5V power supply and a RS422 twisted pair serial channel connected in parallel bus mode. Thanks to the address selector micro-switch located at each IO board, the header can send data and poll information from every board connected to the bus.

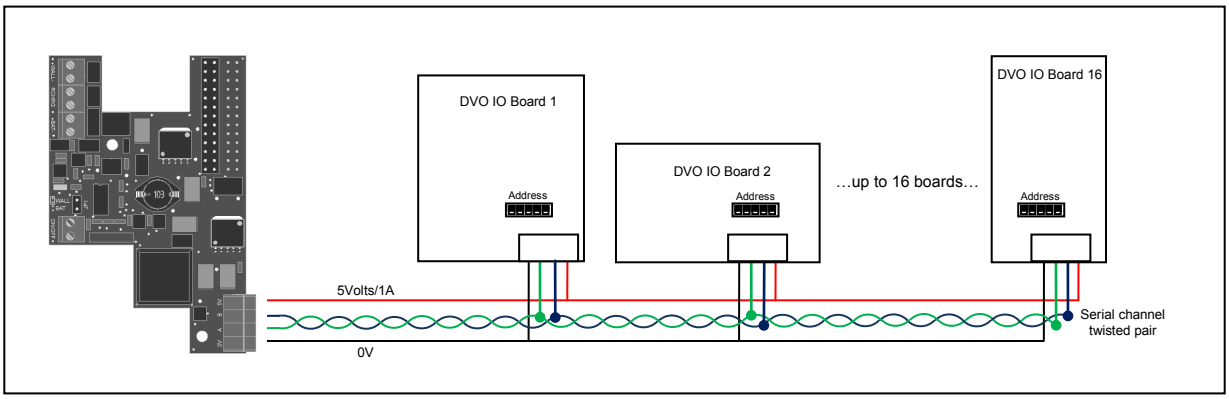



**Fig. 4 DVO IO expansion board bus**

After the First Wake Up state, the Header1 control unit executes the initial discovering process by individually polling to all possible address. It takes roughly one second and, once it is finished, the Header1 board knows how many boards are connected and what their address are.

User programs can read/write data from/to boards thanks to *dv0_manager* program which acts as a tunnel between user programs and IO boards. A complete and easy to use library is free available for C/C++, Java and Python. See Section 4 for more information and examples.

So, user programs connect to *dvo_manager*, which talks with the Header1 control unit, the actual responsible for send and receive data through the serial channel. During idle states, that is, when user programs don't invoke any library function, the Header1 control unit continuously polls to all known board to get the value of their inputs and maintains a local database with these values. Thanks to this database, user programs functions that want to read board's input values exhibit no extra delay independently of how many boards are connected.

Also, user program functions that want to write to a specific board are considered as a high priority procedure by the Heade1 control unit, and it only wait the current poll operation to launch the command to the specified board. This strategy ensures a maximum latency of 15 ms both for read and write operations from user programs.

### 1.2.1 Discovery process

The broad discovery process, that is, a sequential enquiry to all possible 63 address, is made whenever:

- The First Wake up occurs
- Every time that the Header1 goes to On State from Idle State

An individual discovery process, that is, an enquire to a unique and specific address, is made whenever a user program wants to access to a board that is not present in the Header1 control unit database. In order to maintain the latency time declared above, the control unit returns an error as "Board not present" to the user program function, but simultaneously sends a poll to this board. If this board is really connected, the next user program function to this board will be successful.

This allows a "hot connection" feature for the DV0 Bus but the user program procedure has to check every function return value and in case of "Board not present" error, retry the function call instead of treating this error as an exception.

## 2. ELECTRICAL CHARACTERISTICS

This section explains the electrical characteristics of each Header1 connector and the length issues related to the DV0 bus cabling.

### 2.1 Identifying connectors

The Figure 5 shows the location of all connectors of Header1 board

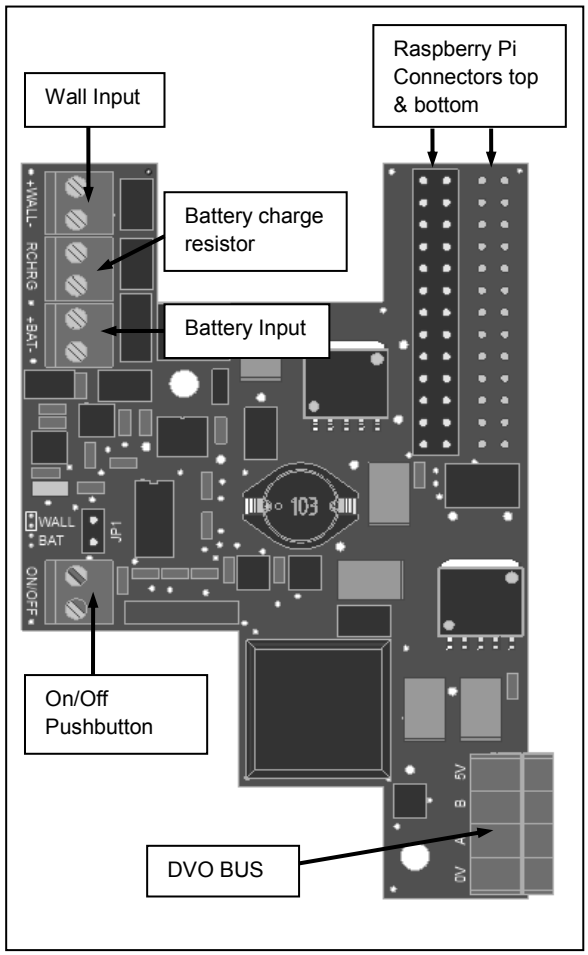



**Fig. 5 Header1 connectors**

#### 2.1.1 Wall input

Main Input power for generating Raspberry Pi 5V, DV0 Bus 5V and battery charge. Connect the positive wire to the screw marked as "+" and the negative wire to the screw marked as "-".

Maximum Voltage……..27V
Minimum Voltage……..6.5V
Reverse polarity protection……Yes
Solid/Stranded wire……AWG 16 to AWG 24
Wire cross section ……1.5mm to 0.2mm

See 2.3 Power Consumption for input current calculation.

#### 2.1.2 Battery charge resistor

This connector is intended for increase the battery charge current by connecting an external resistor. See 3.4 Battery dimensioning and charging.

#### 2.1.3 Battery input

Secondary input power for generating Raspberry Pi 5V, DV0 Bus 5V and battery charge. Connect the positive wire to the screw marked as "+" and the negative wire to the screw marked as "-".

Maximum Voltage……..27V
Minimum Voltage……6V
Reverse polarity protection……..Yes
Solid/Stranded wire……..…AWG 16 to AWG 24
Wire cross section ……1.5mm to 0.2mm

See 2.3 Power Consumption for input current calculation.

#### 2.1.4 On/Off Pushbutton

A momentary pushbutton connected to this input allows to rise the 5V power for the Raspberry Pi and the DV0 Bus connecter if it is pressed for more than 2 seconds. Also, a Power Off Process should be initiated if the button is pressed for more than 4 seconds.

The operation of this input is disabled if the *dv0_manager* program is called with the command line parameter *–no_push_active.*

This input has an internal 10K pull up resistor connected to the 3.3 V Header1 rail. If a device other than a pushbutton is connected to this input, keep in mind that the Header1 control unit requires a voltage below 660mV, so a minimum of 300µA sink current needs to be assured. The screw aligned with the text "OFF" is connected to the common ground: Wall and Battery "-" inputs and 0V of DV0 Bus.

No high voltage transients protection are placed so it is not recommended to assert a large cable between this input and the pushbutton if this cable can be placed in parallel with noisy or high voltage power lines.

### 2.1.5 DV0 Bus connector

This receptacle connector contains power and data for IO expansion board. An extracting socket with screws is supplied with the Header1 board to facilitate the installation of IO expansion board bus cabling (AWG 16..24, cross section 1.5 to 0.2 mm)

Technical data:

- Socket reference: 20020004-D041B01LF from FCI.
- Solid/Stranded wire: AWG 16 to AWG 26
- Wire cross section: 1.5mm to 0.2mm
- Data signal A and B ESD, EFT and Surge protection: IEC 61000-4-2 (ESD), IEC 61000-4-4 (EFT) and IEC 61000-4-5 (Surge)
- 5V output maximum current: 1A
- 5V output short-circuit current: 2A

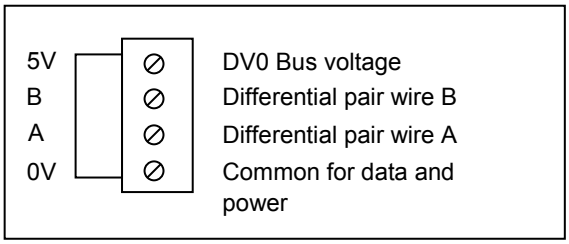



**Fig. 7 DVO Bus connector pin out**

Connect differential pair A to all A inputs of connected IO boards, and pair B to all B inputs as well. Also, connect the common data pin (0V) to all IO boards. The output marked as 5V is the DV0 bus voltage output. It is raised to 5V during the Power On Process and disconnected during the Power Off Process (see 1.1.5 and 1.1.3). This output can be used to fed the IO boards connected to the bus, but it is not mandatory since IO boards can be locally supplied, as explained at 2.4.2 Voltage drop.

All IO Boards PCB are marked with the legend shown at Figure 7.

## 2.2 Absolute maximum ratings

Absolute maximum ratings for the Header1 board are listed below. Exposure to these maximum rating conditions for extended periods may affect device reliability. Functional operation of the device at these, or any other conditions above the parameters indicated in the operation listings of this specification, is not assured.

*Operating Ambient temperature……....0°C to +70°C*
*Storage temperature .....-55°C to +125°C*
*Voltage on Wall Input.....-28V to +28.0V*
*Voltage on Battery Input.....-28V to +28.0V*
*Voltage on Pushbutton connector.....-0.3V to +3.3V*
*Maximum Current out to Raspberry Pi©……..…..1A*
*Maximum Current out to DV0 Bus……..…..1.2A*

Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at those or any other conditions above those indicated in the operation listings of this specification is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability.

## 2.3 Power Consumption

Either in Wall Mode or Battery Mode, the power flows from the Wall input or the Battery input depending only on what input has the higher level of voltage. Because of a diode connection, when the higher input falls below the lower, the Header1 softly balances the path of power so that the output to the Raspberry Pi, the DVO Bus and the internal power doesn't note the change.

Figure 7 shows a simplified diagram of power path.

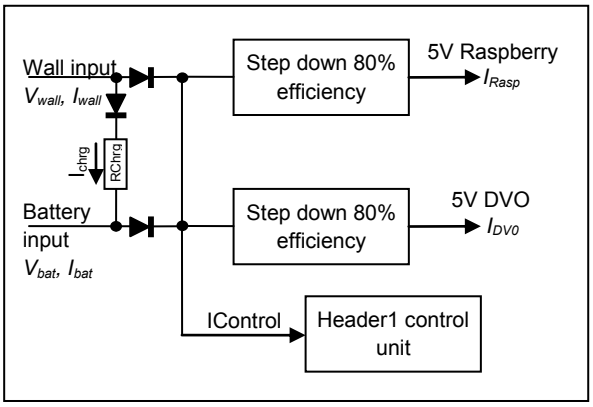



**Fig. 7 Power path**

The power consumption is the product of voltage and current so, given a nominal voltage for wall or battery input as a design decision, the input current has to be calculated for dimensioning the wall input adapter and the capacity of the battery.

### 2.3.1 Wall input consumption

To do that, lets begin with the total power equation of Wall input:

$$P_{wall} = P_{rasp}+P_{DV0}+P_{control}+P_{diode}+P_{charge}$$

$P_{rasp}$ is the power delivered to the Raspberry Pi plus the amount of losses due to the inefficiency of the step down voltage regulators. As its efficiency is, roughly, 80 per cert, $P_{rasp}$ can be expressed as

$$P_{rasp} = 5 \times I_{Rasp} / 0.8$$

The same reasoning applies for PDV0

$$P_{DVO} = 5 \times I_{DVO} / 0.8$$

$P_{control}$ is the power that needs the Header1 control unit to operate and is quite different whether this control unit is in the sleep state or in the off state. During on state, the current consumption is 25mA while in the sleep state this current falls to 3.5mA. The control unit voltage regulator is a linear regulator so the current delivered is constant, which yields a power consumption of

$$P_{control} = V_{wall} \times I_{control}$$

$P_{diode}$ is the loss of power due to the balancing diodes and can be computed as the sum of the input current to $V_{Rasp}$ and $V_{DVO}$ step down regulators and the control current, all multiplied by 0.4 as the diode nominal voltage drop.

$$P_{diode} = 0.4(P_{rasp}/V_{wall} + P_{DV0}/V_{wall} + I_{control})$$

Finally, $P_{charge}$ depends on the difference of voltage between Wall input and Battery input and the charging resistor. Assuming a diode voltage drop of 0.2V, this power can be computed as:

$$P_{charge} = V_{wall}(V_{wall} - V_{bat} - 0.2)/R_{chrg}$$

The charge resistor depends on the capacity of the battery and the time required to obtain a full charge condition. If no external resistor is applied, the charge resistor value is 470Ω

### 2.3.2 Battery input consumption

The power equation of Battery input is only slightly different of the Wall input in the sense of that no power charging is involved:

$$P_{battery} = P_{rasp}+P_{DV0}+P_{control}+P_{diode}$$

And the expressions for $P_{control}$ and $P_{diode}$ are:

$$P_{control} = V_{bat} \times I_{control}$$

$$P_{diode} = 0.4(P_{rasp}/V_{bat} + P_{DV0}/V_{bat} + I_{control})$$

Note that the power needed to feed the Header1 control unit is proportional to the battery voltage so, in terms of battery duration, the lower the battery voltage, the better the battery performance.

### 2.3.3 Power calculation examples

In the formula expressed above, there are parameters that are design decisions, as Wall input voltage and Battery input voltage, some constants like $I_{control}$ or $R_{chrg}$ and some parameters that can be estimate like $I_{Rasp}$ and $I_{DVO.}$

The target is to obtain $I_{wall}$ and $I_{bat}$ to calculate the power of the Wall adapter and the capacity of the battery.

The value of constants parameters are:

$I_{control}$*(On state)……..……..0.025A*
$I_{control}$ *(Off state)…….……..0.0035A*
$R_{chrg}$ *(no external resistor)……470Ω*

As for estimated parameters, $I_{DVO}$ is obtained as the sum of all boards current, easily available in the data sheet of every board. On the other hand, $I_{Rasp}$ is quite difficult to estimate because it is function of how many USB gadgets are connected, Ethernet traffic (if exists) camera consumption, Linux distro, etc.

However, it is possible to estimate the average current consumption of $I_{Rasp}$ assuming the following data (source, official Raspberry site):

*Raspberry Pi© A nominal consumption……....0.5A*
*Raspberry Pi© B nominal consumption……....0.7A*
*Camera consumption……..……..0.2A*

And for each gadget connected to USB, try to find out their power consumption (in Watts) and divide it by 5 to obtain the current consumption (in Amperes) and add it to the value of $I_{Rasp}$

Note that all values have to be expressed in Volts, Amperes and Watts to avoid magnitude

Following examples will be used for power calculation and for battery

**Example 1.**

Set Wall input to 15V, Battery input to 12V, no external charge resistors, Raspberry Pi A model with no camera and two IO_A boards connected to the bus.

Lets begin with the on state power consumption. From the data described above, calculate $P_{Rasp}$:

$$P_{rasp} = 5 \times 0.5 / 0.8 = 3.12 \quad [W]$$

From IO_A datasheet, find out the maximum current consumption (0.134A) and calculate $P_{DVO}$

$$P_{DVO} = 5 \times (0.134 + 0.134) / 0.8 = 1.675 \quad [W]$$

Continue with the control unit power, using the greater voltage among Wall or Battery input. In normal operation conditions, voltage at Wall input is greater than Battery input. Only in the absence of Wall power will be the Battery input get the greater value.

$$P_{control} = V_{wall} \times I_{control} = 15 \times 0.025 = 0.375 \quad [W]$$

Then, compute $P_{diode}$

$$P_{diode} = 0.4(P_{rasp}/V_{wall} + P_{DV0}/V_{wall} + I_{control}) = 0.4(3.12/15+1.675/15+0.025) = 0.138 \text{ [W]}$$

Follow with $P_{charge}$:

$$P_{charge} = V_{wall}(V_{wall} - V_{bat} - 0.2)/R_{chrg} = 15(15-12-0.2)/470 = 0.006 \text{ [W]}$$

And finally, add altogether to carry out the total power

$$P_{wall} = P_{rasp}+P_{DV0}+P_{control}+P_{diode}+P_{charge} = 3.12 + 1.675 + 0.375 + 0.138 + 0.006 = 5.31 \quad [W]$$

With this information it is possible to find a suitable wall adapter but sometimes, adapters are rated by their output current instead of their output power. In this case, simply divide the calculated power by the nominal voltage:

$$I_{wall} = P_{wall} / V_{wall} = 5.31/15 = 0.354 \text{ [A]}$$

As for the battery, it is important to know the power consumption during both the on state and the off state.
During the on state, battery power is

$$P_{battery(ON)} = P_{rasp}+P_{DV0}+P_{control}+P_{diode}$$

where

$$P_{control(ON)} = V_{bat} \times I_{control(ON)} = 12 \times 0.025 = 0.3 \quad [W]$$

$$P_{diode(ON)} = 0.4(P_{rasp}/V_{bat} + P_{DV0}/V_{bat} + I_{control(ON)}) = 0.4(3.12/12+1.675/12+0.025) = 0.170 \quad [W]$$

then,

$$P_{battery(ON)} = P_{rasp}+P_{DV0}+P_{control(ON)}+P_{diode(ON)} = 3.12 + 1.675 + 0.3 + 0.17 = 5.26 \quad [W]$$

and dividing by $V_{bat}$ to obtain $I_{bat}$ current yields

$$I_{bat(ON)} = P_{battery(ON)}/12 = 0.439 \text{ [A]}$$

During the off state, battery power is

$$P_{battery(OFF)} = P_{control(OFF)}+P_{diode(OFF)}$$

because no energy is supply to the board or the bus. The power loss on the diode is

$$P_{diode(OFF)} = 0.4 \times I_{control(OFF)} = 0.4 \times 0.0035 = 0.0014 \text{ [W]}$$

*and power control is reduced to*

$$P_{control(OFF)} = V_{bat} \times I_{control(OFF)} = 12 \times 0.0035 = 0.042 \text{ [W]}$$

Total power is

$$P_{battery(OFF)} = P_{control(OFF)}+P_{diode(OFF)} = 0.042 + 0.0014 = 0.043 \text{ [W]}$$

Dividing by $V_{bat}$ to obtain $I_{bat}$ current:

$$I_{bat(OFF)} = P_{battery(OFF)}/12 = 0.0036 \text{ [A]}$$

Dealing with batteries, it is usual to manage nominal voltage and discharge current as a design parameters to compute the battery size and estimations on battery duration and charging time as well. Both $I_{Bat(OFF)}$ and $I_{bat(ON)}$ *will* be used at 3. Battery dimensioning.

**Example 2.**

Set Wall input to 12V, Battery input to 7.2V, no external charge resistors, Raspberry Pi B model with camera, a WiFi dongle which has a rated power of 200mW, and four IO_A boards connected to the bus.

The current drown by the Raspberry Pi in this case is

$$I_{rasp} = 0.7 + 0.2 + 0.2/5 = 0.940 \text{ [A]}$$

and the power

$$P_{rasp} = 5 \times 0.940 / 0.8 = 5.87 \text{ [W]}$$

Using the same sources of information that on the example 1, it is easy to find

$$P_{DVO} = 5 \times (4 \times 0.134) ) / 0.8 = 3.35 \text{ [W]}$$

$$P_{control} = V_{wall} \times I_{control} = 12 \times 0.025 = 0.3 \text{ [W]}$$

$$P_{diode} = 0.4(P_{rasp}/V_{wall} + P_{DV0}/V_{wall} + I_{control}) = 0.4(5.87/12+3.35/12+0.025) = 0.317 \text{ [W]}$$

$$P_{charge} = V_{wall} (V_{wall} - V_{bat} - 0.2)/R_{chrg} = = 12(12-7.2-0.2)/470 = 0.006 \text{ [W]}$$

And finally

$$P_{wall} = P_{rasp}+P_{DV0}+P_{control}+P_{diode}+P_{charge} = 5.87 + 3.35 +0.3 + 0.317 + 0.006 = 9.84 \text{ [W]}$$

$$I_{wall} = P_{wall} / V_{wall} = 9.84/12 = 0.820 \text{ [A]}$$

As for the battery, during the on state, the intermediate values are

$$P_{control(ON)} = V_{bat} \times I_{control(ON)} = 7.2 \times 0.025 = 0.180 \text{ [W]}$$

$$P_{diode(ON)} = 0.4(P_{rasp}/V_{bat} + P_{DV0}/V_{bat} + I_{control(ON)}) = 0.4(5.87/7.2+3.35/7.2+0.025) =0.522 \text{ [W]}$$

then,

$$P_{battery(ON)} = P_{rasp}+P_{DV0}+P_{control(ON)}+P_{diode(ON)} = = 5.87+ 3.35 + 0.18 +0.522 = 9.92 \text{ [W]}$$

and dividing by $V_{bat}$ to obtain $I_{bat}$ current yields

$$I_{bat(ON)} = P_{battery(ON)}/7.2 = 1.38 \text{ [A]}$$

During the off state, the intermediate values are

$$P_{diode(OFF)} = 0.4 \times I_{control(OFF)} = 0.4 \times 0.0035 = 0.0014 \text{ [W]}$$

$$P_{control(OFF)} = V_{bat} \times I_{control(OFF)} = 7.2 \times 0.0035 = 0.025 \text{ [W]}$$

*then*

$$P_{battery(OFF)} = P_{control(OFF)}+P_{diode(OFF)} = 0.025 + 0.0014 = 0.026 \text{ [W]}$$

Dividing by $V_{bat}$ to obtain $I_{bat}$ current:

$$I_{bat(OFF)} = P_{battery(OFF)}/7.2 = 0.0036 \text{ [A]}$$

Note that although the $I_{bat(OFF)}$ of example 1 and example 2 are similar, the power of example 2 is reduced proportionally to the reduction of battery nominal voltage. So, to maximize the duration of the battery during off state, it is better to choice a low voltage battery.

## 2.4 DVO Bus length

There are two different considerations about DV0 Bus wire length. On one hand, there is the data transmission degrading, on the other hand there is the issue of the voltage drop due to the resistance of the cable.

### 2.4.1 Data transmission degrading

Data is transmitted using a RS422 standard level of differential voltage at 100kbps on the wires marked as A and B at the DV0 Bus connector. This standard doesn't define a wire type to be used, in opposition to other standards like Ethernet or similar. For this reason there is not an official maximum cable length for this standard.

However, there are a conservative data based on empirical test that assures a 1000 meters length at 100 kbsp using a twisted pair of 15pf, 5ns/m of propagation speed.

To achieve the maximum noise rejection that RS422 offers, it is mandatory to use a twisted pair. We recommend a twisted pair of AWG 20 (0.5 $mm^2$) because test realized with this cable had shown no degradation over 50 meters bus with 16 IO boards connected (no termination resistors are required for that distance)

Even with the CRC that protects all frames which travel through the bus and despite the excellent performance shown over 50 meters bus length, we don't recommend to exceed this limit due to the ground common mode noise that could appear if the power wires of bus are not strong enough or there is a ground loop.

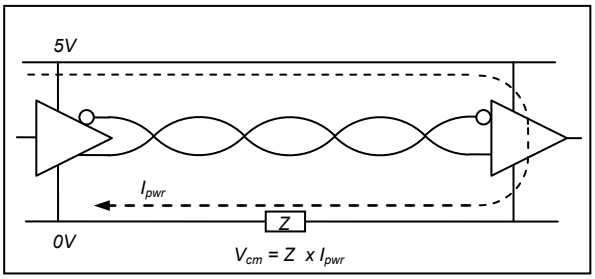



**Fig. 8 Common mode noise**

The common mode noise is generated by the power current when it flows throughout the power wires. As the power wires don't have a zero impedance, a common voltage appears (see Figure 8) and, if this noise is greater than 7 volts, the receiver can't decode data so the current frame will be ignored. Also, all strong electric or magnetic field close to the power wires can induce a common mode voltage as well.

To reduce the common mode voltage noise, it is strongly recommended:

- Twist the power wires as the data wires. This reduces the inductive factor of impedance and the magnetic fields disturbance.
- Avoid cabling power wires in parallel with other power lines, specially if they carry strong inductive loads, as motors or electro valves.

### 2.4.2 Voltage drop

The voltage drop across the power wires is due to the DC impedance (i.e., the resistance) of these wires. It is easy to calculate this voltage drop average value knowing the section of the cable, its length and the average current consumption of the IO board.

Knowing the section of the cable, manufacturers report its resistivity ($\rho$) in ohms/meter. Then, to obtain the voltage drop

$$V_{drop} = I_{avg} \times length \times \rho$$

Using the example 2 explained above, with AWG 20 section and 25 meters length between the Header1 board and the IO board, the voltage drop will be

$$V_{drop} = I_{avg} \times length \times \rho = 4 \times 0.134 \times 25 \times 0.033 = 0.442\ [V]$$

In the above expression, the figure 0.033 is the nominal resistivity of AWG 20 wire (from manufacturer). As all DVO IO boards can withstand this voltage drop over 5V, this example is correct. However, this voltage drop is near the limits so, if we plan a larger bus length, a new approach is needed.

To solve this issue, there are two solutions available:

- Increase the wire section
- Feed IO board with local power supplies

Increasing the wire section is the simplest solution but sometimes is not practical due to the cost increment and the difficulty of cabling thick wires on the small connectors of IO boards.

The use of local power supplies offers more flexibility and solves the problem of connecting heavy loads (as motors or servos) to the DV0 Bus power.

Figure 9 shows this 3-wires approach. When using local powering, no high current flows through the ground line but there is the risk of creating a ground loop due to the difference of potential between the earth reference of the local power supplies. However, the common mode voltage generated in this case is far lower than the one due to power supply current.

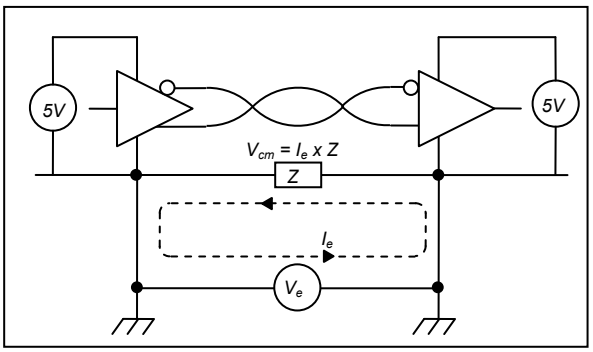



**Fig. 9 Local powering and ground loop**

**CAUTION**:

Header1 and DV0 IO boards are protected against high transient voltage according to IEC 61000-4-5 but it is not recommended to deploy a large length of cable with different earths references: lighting can generate a sudden high energy transient that would destroy the transceivers despite the surge protections.

This approach is intended just to avoid large currents flowing through the power lines, not to increase the distance between Header1 and IO board. If you want to, do it at your risk, but at least add a more powerful TVS protection and assure a good earth connection at both power supplies. Also, it is strongly recommended to avoid placing data lines close to power lines.

## 3. BATTERY DIMENSIONING

The Header1 board is intended for two battery scenarios:

- **Wall Mode**: The system is powered by a wall adapter and the battery is dimensioned only for assuring a clean shutdown.
- **Battery Mode**: The system is mainly powered by a wall adapter but in case of failure, the system continues its operation powered by the battery until the voltage battery drops below a save preconfigured value witch allows a clean shutdown

Obviously, the second scenario requires a much powerful battery than the first one.

Following in this section, an explanation of how to estimate the battery capacity will be given, along with some discussions on the most suitable types of batteries and their nominal voltage. Also, it is important to calculate the optimum value for the charging resistor. However, prior of battery calculation, it is worth to comment some issues related to battery chemistry and capacity.

### 3.1 Battery considerations

In this document, all references to "battery" stand for "rechargeable battery". For our purposes, rechargeable batteries are defined by three parameters:

- **Chemistry**: Can be Nickel based, Lead based or Lithium based
- **Capacity**: Indicates the amount of energy that the battery can store
- **Internal resistance**: The great the internal resistance, the lower the current that can be drown from the battery.

#### 3.1.1 Battery chemistry

Lithium base batteries requires a smart charge system. The Header1 board **DOESN'T HAVE A SMART CHARGE SYSTEM**, so Lithium based batteries are **FORBIDEN** to Header1.

**CAUTION**:

Some Lithium batteries have internal protection against over and under voltage, but most batteries don't have any kind of such protection so the risk of blowing or firing a Lithium based battery is extremely high if it is connected to the Header1 board

Nickel based batteries are, nowadays, only Nickel-Metal Hydride chemistry because the ancient Nickel Cadmium technology has become prohibited. This batteries are robust, easy to charge if the current is not great, better ratio capacity/weight and capacity/volume than lead batteries but much expensive.

Lead battery are also robust and easy to charge, cheaper than Nickel batteries, but they need much volume and their weight is important.

As a rule of thumb, when in battery mode, if there is no restriction about volume and weight, the better decision is a lead battery. More exactly, we recommend sealed lead acid because they are maintenance-free and their price is very attractive.

However, during the last years, NiMH batteries have decreased their price, close to the lead battery type, so it could be interesting to choice a NiMH if the weight-volume restrictions are important. Also, when in wall mode, there is no need of powerful battery and it is better to choice a little 9V, PP3, NiMH model, as explained below at 3.2 Wall Mode example.

The most important parameter related to the chemistry is the cell voltage at full charge, nominal and end of charge. The voltage per cell of both chemistries exhibits the same type of curve as they are discharge.

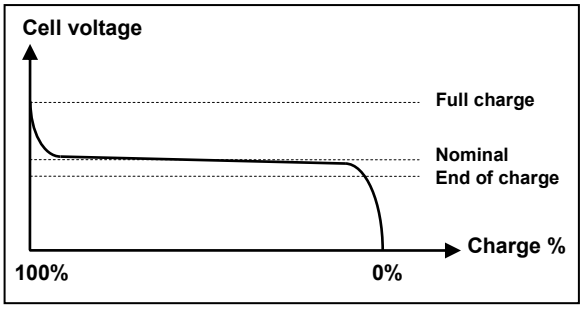



**Fig. 10 NiMH and Lead battery discharge curve**

Figure 10 shows the discharge curve. At full charge the cell voltage is slightly higher than the nominal value, but this level drops early as the battery discharge begin. Then, there is a plateau at nominal voltage that remains for the 90% of the discharge cycle. Finally, close to the end of charge, there is an abrupt drop of voltage. It is important that when the battery reaches the end of voltage threshold, the operating systems issues a shutdown immediately because the time for a full voltage drop is unpredictable. Also, its is recommended to be conservative with this threshold.

Table 1 shows the standard values for full charge, nominal and end of charge values per cell and per chemistry.

Commercial battery packs come with some standard number of cells. Designers have to multiply the number of cell by the End of Charge value to obtain the *min_voltage* parameter. The *dv0_manager* program continuously compares the battery voltage against the *min_voltage* parameter and sends a shutdown command when the battery voltage falls below this value.

**Table 1. Cell voltages**

| Chemistry | Full charge | Nominal | End of charge |
|---|---|---|---|
| NiMH | 1.8V | 1.2V | 0.9V |
| Lead Acid | 2.1V | *2V* | 1.95V |

Caution: check these values with the manufacturer data because there can be a significant tolerance among models and types.

The default value for *min_voltage* is 6V, but can be modified as a command line parameter when launching *dv0_manager*.

Table 2 shows the recommended values of *min_voltage* for the most common packs of commercial batteries

**Table 2. min_voltage recommended values**

| Chemistry | Nominal voltage | Number of cells | *min_voltage* value |
|---|---|---|---|
| Lead Acid | 6V | 3 | Not recommended |
| Lead Acid | 12V | 6 | 11.9 |
| Lead Acid | 24V | 12 | |
| NiMH | 7.2V | 6 | 6 |
| NiMH | 8.4V | 7 | 7 |
| NiMH | 9.6V | 8 | 8 |
| NiMH | 14.4V | 12 | 12 |
| NiMH | 18V | 15 | 15 |

### 3.1.2 Battery capacity

In theory, battery capacity should express the maximum amount of energy that the battery can store, but in practice, it is expressed as Amperes-Hour (Ah), which is a charge magnitude, not an energy magnitude. Designers have to multiply the "capacity" (Ah) by the nominal voltage battery pack (Volts) to obtain Watts-Hour which it is an energy magnitude.

However, despite the formal considerations, it is quite convenient to use the battery capacity in Ampere-Hour to estimate the battery duration. For example, a battery of 4800mAh, as a rule of thumb, can supply a system that requires 4.8A during one hour, or a system that requires 2.4A for two hours and so forth.

Also, the battery capacity is used as a relative measure of charging or discharging current. For example, if a battery of 4800mAh is discharged with a current of 4.8 amperes, it is said that the discharge current is 1C, if the current is 9.6A then the current is 2C, and for a 2.4A, the current is C/2. This value will be used for charging current calculation below, at 3.4 Battery dimensioning examples.

### 3.1.3 Internal resistance

The internal resistance depends on the quality of the battery and determines its application. The greater the internal resistance, the greater the voltage drop for a given current. Batteries designed to supply motors have a very low resistance and allows discharge current as big as 30C o more.

For the purposes of this application, the current consumption involved is quite low so no issues at this point are expected, except for the little 9V battery in the example 3.4.1

## 3.2 Wall Mode

**Note: Wall Mode operation is selected by leaving connected the jumper 1, as shown in Figure 2.**

The Wall Mode is intended for applications where the main power source comes from a wall adapter supply and a battery is needed only for safe shutdown purposes or sleeping features as time keeping, scheduled restart, on/off pushbutton, etc.

**Shutdown condition**: The Header1 control unit issues a shutdown command to the Raspberry Pi board whenever the voltage on the Wall Input is lower than the voltage in the Battery Input for more than five seconds. When the shutdown starts, the control unit of the Header1 waits a determinate number of seconds (*shutdown_time*) and cuts the power of the Raspberry Pi and the DV0 bus. It is possible and recommended to cut the power of the DV0 bus by setting the parameter *–cut_vbus* as a command line parameter of *dv0_manager.*

**Energy needs**: The battery must supply energy to the Raspberry Board during the shutdown process and to the Header1 control unit to keep the real time clock running.

### 3.2.1 Battery capacity calculation procedure

To calculate the battery capacity, designers must follows these steps:

- Determine $P_{battery}$ from 2.3.2. Set $P_{DV0}$ zero if parameter –cut_vbus is set
- Obtain $I_{control(Off)}$ *from 2.3.3*

- Measure the actual shutdown time and set it as a command line parameter of *dv0_manager* by setting *–shutdown_time nn* where *nn* is the number of seconds of shutdown time measured. Default parameter is 30, but experiences proof that the Raspbian distro needs only 15 seconds to completely halt. Let's call it $T_{sd}$ (in seconds)
- Decide how many hours must the Header1 kept the real time clock running before battery fails. Let's call it $T_{rtc}$ (in hours)

Then, for a given battery with nominal voltage $V_n$, the capacity *C* (in Amperes x Hour) required is

$$C = \frac{(P_{battery} \cdot T_{sd})/3600 + V_n \cdot I_{control(Off)} \cdot T_{rtc}}{V_n} \quad [Ah]$$

Note: if the charge/discharge cycle is periodic, it is not recommended to dimension the battery to the exact value found above, since this means that a full discharge state is achieve each cycle. Keep in mind that batterie's life is rated to rough 500 full discharge cycles but this value increases dramatically as the discharge level increases.

### 3.2.2 Battery charge calculation procedure

Because there is no temperature or voltage sensor inside the Header1 board, charging battery is made by supplying a low rate continuous current throughout a resistor from Wall input to the battery. There is a internal resistor of 470Ω rated at 1W of power dissipation for this purpose.

**CAUTION**:

Maximum charge current for this method is C/4. However, we recommend at maximum current of C/10 for completely safe operation an wide battery life

Designers must calculate the theoretical value of the charge resistor ($R_{chrg}$) and, if it is close to the internal resistor of 470Ω, no more actions are needed but, if the final result is lower than 470Ω then, an external resistor must be place so that the parallel value of the external resistor with the internal one matches the theoretical value.

These are the steps:

- Decide the time, in hours, required to full charge the battery from a full discharged state. Suitable values are form 10 to 16. Let's call it $T_{chrg}$ (in hours).
- Calculate the charge current for a giving battery capacity *C* (in Amperes x hour) computed as $I_{chrg} = C/T_{chrg}$
- Decide a voltage for Wall input. The closer to the battery voltage, the better. Let's call it $V_{wall.}$
- Calculate $R_{chrg}$ as:

$$R_{chrg} = (V_{WALL}-V_n)/I_{chrg}$$

- Calculate the power dissipated by this resistor as:

$$P_{Rchrg} = R_{chrg} * (I_{chrg})^2$$

Then, compares $R_{chrg}$ and $P_{Rchrg}$ with the internal value of 470Ω, 1W. It they are close enough, just let the internal resistor charge the battery and recalculate the charging time (in hours) as

$$T_{chrg} = C / (V_{wall}-V_n) / 470$$

If $R_{chrg}$ is quite lower than 470Ω, then connect an external resistor ($R_{ext}$) to the battery charge connector (see 2.1) so that

$$R_{ext} = (470 \cdot R_{chrg})/470-R_{chrg})$$

Which is the solution of the parallel resistor equation. Maintain the rated power calculated for $R_{chrg}$

## 3.3 Battery mode

**Note: Battery Mode operation is selected by leaving unconnected the jumper 1, as shown in Figure 2.**

The Battery Mode is intended for applications where the main power source is a battery, and Wall Input is used just for slow battery charging, saving battery energy or for feeding the board when the battery needs to be temporally disconnected.

The 5V power to Rasperry Pi is always obtained from the input source that exhibits the upper voltage value and automatic balance is made when one source falls below the other one without affecting the stability of Raspberry Pi power supply.

**Shutdown condition**: the shutdown command is issued only when the Battery input voltage falls below *min_voltage* for more than five seconds, no matters what voltage is at Wall input. As in Wall mode, it is possible to cut the power of the DV0 bus setting the parameter *–cut_vbus* as a command line parameter of *dv0_manager*.

**Energy needs**: The battery must supply energy to the Raspberry Board and DV0 bus during the absence of wall input source. Also, if the designer plans to keep the system in stand by and start it at scheduled time, the battery must supply energy to the Header1 control unit to maintain the real time clock and the restart scheduler.

### 3.3.1 Battery capacity calculation procedure

To calculate the battery capacity, designers must follows these steps:

- Determine $P_{battery}$ from 2.3.2.
- Obtain $I_{control(Off)}$ *from 2.3.3*
- Decide how many time (in hours) must the full system (Raspberry plus DV0 boards running) be powered by the battery (that is, no Wall input energy). Let's call it $T_{on}$.
- Decide how many time (in hours) must the system be in off state (that is, Raspberry and DV0 boards off and Header1 control unit keeps running the real time clock) powered only by the battery. Let's call it $T_{off.}$

Then, for a given battery with nominal voltage $V_n$, the capacity $C$ (in Amperes x Hour) required is

$$C = P_{battery} \cdot T_{on} / V_n + I_{control(Off)} * T_{off}$$

### 3.3.2 Battery charge calculation procedure

There is no methodological difference between wall mode and battery mode for charge calculations purposes, just follow the same steps described at 3.2.2. However, it is worth to note that the power required for the charge resistor will be much higher that 1W, so the internal resistance of the Header1 is useless in this mode and an external resistor will be mandatory.

## 3.4 Battery dimensioning examples

The following examples cover the mainly intended use of Header1 board:

- Clean shutdown to avoid SD card corruption (wall mode)
- Control and surveillance systems (battery mode)

### 3.4.1 Wall mode calculations example

**Case**: the system is a panel information in a mall. It is supplied by a wall input source form 8.00 am. to 10.00 pm. and has Raspberry Pi© A model with no camera and two IO_A boards connected to the bus. There is no need to save the state of IO_A boards, so –*cut_vbus* parameter is set when invoking *dv0_manager*.

At 8.00, the power at Wall input is present and the Header1 will start the Raspberry and the DV0 bus boards. It is required that the Header1 sets the real time clock date (kept during all night long) to the Raspberry operating system so –*force_date* parameter is set when invoking *dv0_manager*.

Assuming that the display is powered by an external source, and using the Example 1 from 2.3.3, we know that the power of Wall input is $P_{wall}$ *= 5.31 [W]*. Also from 2.3.3, the power consumption during shutdown time (when wall input disappears) is $P_{battery(ON)}$ *= 5.26 - 1.675 = 3.575 [W]* (because *cut_vbus* is set).

$I_{control(Off)}$ *is 0.0035A (from 2.3.3)*

Several measuring shutdown time gives a mean time of 12 seconds. Let's be conservative and set $T_{sd}$ as 20 seconds so *dv0_manager* is called with –*shutdown_time 20*.

Battery has to survive during 10.00 pm to 8.00 am so $T_{rtc}$ is 10.

A small 9V, PP3, NiMH seems to be the better choice to this scenario. Only its capacity has to be calculate to full set battery requirements.

**NOTE**: Due to the similitude to common PP3 non rechargeable batteries, witch have 9V of nominal voltage, most distributors declare PP3 NiMH as a 9V battery but, in fact, they can made as seven cells of NiMH, witch yields a 8.4V nominal voltage or eight cells, witch yields 9.6V nominal voltage. Let set $V_n$ = 8.4V, as it's the most common value on the market.

Then the capacity $C$ required is

$$C = \frac{(P_{battery} \cdot T_{sd})/3600 + V_n \cdot I_{control(Off)} \cdot T_{rtc}}{V_n}$$

$$C = \frac{(3.575 \cdot 20)/3600 + 8.4 \cdot 0.0035 \cdot 10}{8.4}$$

$$C = 0.037\ [Ah] = 37\ [mAh]$$

So we can choice a common PP3, 8.4V, 200mA, NiMH battery. Setting $C$=200mAh while discharge $C$ is 37mAh means that the level of discharge is, roughly, 20%. This value is far from 100%, so battery life endurance is mostly improve. Some manufactures publish life estimation depending on discharge depth percent (but some other not)

**NOTE:** Not only $C$ is important to specify the battery. Designers must check the internal resistance value of the selected battery and calculate the voltage drop during shutdown time. Then, knowing that the power delivered by the battery is $P_{battery(ON)}$= 3.575W at 8.4V, we conclude that the current drown from the battery is $I_{bat}$ = 3.575/8.4 = 0.425 [A]. This peak of current, multiplied by the internal battery resistor must ensure that the voltages at Battery input is always greater than 6.5V. That is $R_{bat}$ < (8.4-6.5)/0.435 = 4.36 [Ω]

As for the charge battery, the scenario assures a charging time from 8.00 am to 10.00 pm., that is, 14 hours of charging. The target is to get a full charge from a full discharge state (maybe after holydays) in one day, to assure a all night long supply. Let's try a charging current ratio of C/12. Required charge current is:

$$I_{chrg} = C/T_{chrg} = 0.2/12 = 0.016 \quad [A]$$

For the Wall input voltage, designers must ensure that a full charge battery voltage is always lower that the nominal Wall voltage. Since the selected battery have 7 cells of NiMH and each cell can have a full charge voltage of 1.8V, then, a Wall input greater than 7 x 1.8 = 12V. Let's try a common value of 15V as a nominal voltage for Wall input adapter and then compute $R_{chrg}$ *and* $P_{Rchrg}$:

$$R_{chrg} = (V_{WALL}-V_n)/I_{chrg} = (15-8.4)/0.016 = 412[\Omega]$$
$$P_{Rchrg} = R_{chrg} * (I_{chrg})^2 = 412 *(0.016)^2 = 0.1 \quad [W]$$

Since the internal resistance is 470Ω, rated to 1W, it seems that there are not heat dissipation issues and that the calculated value is so close to the internal value that it is worth to check the behavior of the charge method using only the internal resistance, thus avoiding place any external resistor.

If the charge resistor is just 470Ω, the charge current is $I_{chrg}$ = *(15-8.4)/470 = 0.014 A.* In terms of *C* ratio, that means a charge current of C/14 or, in other words, a full charging time of 14 hours. Just the time allowed for charging, no security factor available.

Designers hate not to have a security factor for their calculus, and dealing with batteries, is a very good practice to left security margins. Fortunately, in this example, a full charge cycle is only needed after a vocational period and, even that, not all charge is need at the end of the day because only a 20% of the total energy is required every night.

Finally, the power required to wall input adapter is obtained from the example 1 of 2.3.3. as $P_{wall}$ = *5.31[W] and a current of* $I_{wall} = P_{wall} / V_{wall}$ = *5.31/15 = 0.354* [A]

**SUMMARY OF REQUIREMENTS:**

- Wall Input adapter: 15V, power > 5.31W, current > 0.354A
- Battery: PP3, 8.4V, 200mA, NiMH
- External charge resistor: none

Software: example of *dv0_manager* call:

```
dv0_manager -cut_vbus -shutdown_time 20
-force_date
```

The cabling diagram is shown at Figure

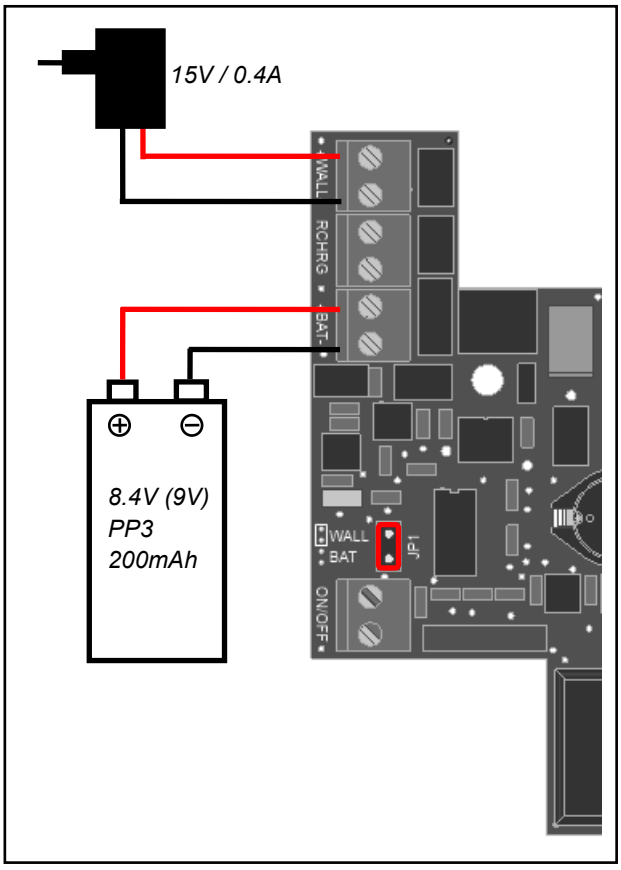



**Fig. 11 Wall mode example**

### 3.4.2 Battery mode calculations examples.

**Case 1**: Night animal movement capture. The system is a Raspberry Pi model A with a infrared camera NoIR (infrared light is supplied by other source). Each night, at 1.00 am the system starts and capture video during 3 hours and it is required to last for a 7 days. No restrictions on weight or size so a 12V Lead Acid battery type is selected.

The battery must supply full energy to the Camera-Raspberry system during 3 hours per day and must fed the Header1 control unit 21 hours to allow starting at 1.00 each day. Software settings are explained below.

From 2.3.3 we obtain that the Raspberry Pi power during on state is

$$P_{rasp} = 5 \times (0.5+0.2) / 0.8 = 4.4 \quad [W]$$

And with a $P_{DV0}$ = 0, the battery power at ON state is

$P_{battery(ON)} = 4.4 + 0.3 + 0.170 = 4.9$ [W]

The battery power at OFF state is, from 2.3.3

$P_{control(OFF)} = V_{bat} \times I_{control(OFF)} = 12 \times 0.0035 = 0.042$ [W]

The capacity per day is

$C(per\ day) = (P_{battery(ON)} \times 3 + P_{control(OFF)} \times 21)/12$ [Ah]

$C(per\ day) = (4.9 \times 3 + 0.042 \times 21)/12 = 1.3$ [Ah]

And multiplying by seven days, we obtain the total capacity:

$C = 1.3 \times 7 = 9.1$ [Ah]

Note: Auto discharge issues have been ignored due to the low period of time considered.

**Software:**

Example of *dv0_manager* call:

```
dv0_manager -force_date -min_voltage 118
-start_up_time 1:00
```

Note that a minimum voltage of 11.8 volts is set, so when the battery charge is close to the end, either a shutdown will be performed or no startup will be launched.

The parameter *start_up_time* tells the *dv0_manager* the periodically starting minute, hour or day (see section 4.3.13 for more details). In this example, the Header1 will start the Raspberry board every day at 1 hour and 0 minutes.

At 4 o'clock, *cron* process must tell to the *dv0_manager* that is time to make a shutdown and go to OFF state. This is achieved either with a *killall* command with the signal power:

```
killall -s SIGPWR dv0_manager
```

or by using the utility *dv0_test* free supplied with *dv0_manager*:

```
dv0_test -power_off
```

Figure 12 shows the battery connection for this case.

**Case 2**: Unassisted meteorological station. The system is a Raspberry Pi A model with camera and two IO boards for weather data acquisition and a 3G USB dongle to send the acquired data and images to a server in internet. This dongle consumes 0.3A at 5V from USB. The station is powered from the mains network but must withstand a lack of mains power for at least 4 hours. It is mandatory that the battery full charge has to be recovered after 8 hours.

Let's suppose that there are a weight or size requirements and choice a 7.2V NiMH battery, together with a Wall adapter of 12V (both are quite common on the shelf values)

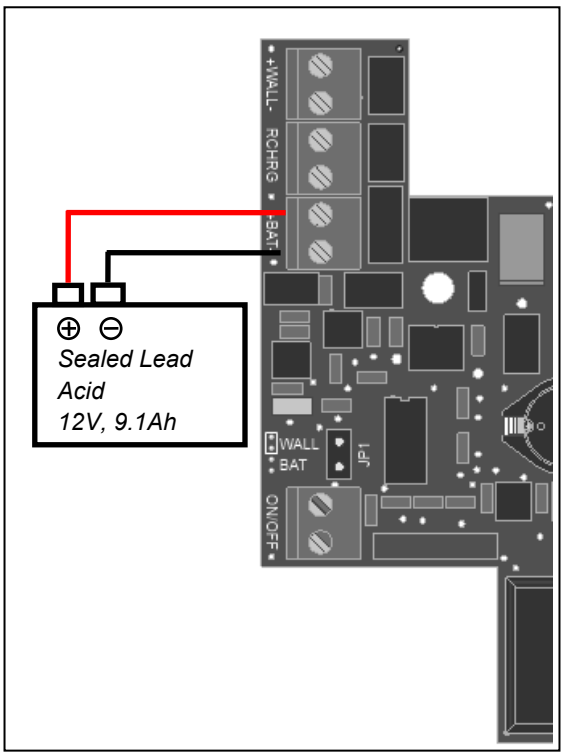



**Fig. 12 Battery mode example Case 1**

From 2.3.3 obtain the Raspberry consumption and add the 3G dongle consumption

$P_{rasp} = 5 \times (0.5+0.2+0.3) / 0.8 = 6.25$ [W]

From 2.3.3. obtain the DV0 consumption

$P_{DVO} = 5 \times (0.134 + 0.134) / 0.8 = 1.675$ [W]

From 2.3.3 and the above calculus compute the power from the battery

$P_{battery(ON)} = P_{rasp}+P_{DV0}+P_{control(ON)}+P_{diode(ON)} =$
$= 6.25 + 1.675 + 0.3 + 0.17 = 8.4$ [W]

Then, calculate the capacity *C* of the battery knowing that $T_{on}$ is set to 4 by the requirements

$C = P_{battery} \cdot T_{on} / V_n = 8.4 \cdot 4 / 7.2 = 4.6$ [Ah]

Because of full charge is required after 6 hours, the charge current must be C/8 so

$I_{chrg} = 4.6 / 8 = 0.575$ [A]

Witch yields a charge resistor of

$R_{chrg} = (V_{wall}-V_n)/I_{chrg} = (12-7.2) / 0.575 = 8$ [Ω]

Because the internal resistor value is 470Ω that is much bigger that 8Ω, there is no need to recalculate the exact parallel value and we set *Rext* as 8Ω. This resistor should dissipate a power of:

$$P_{Rchrg} = R_{chrg} * (I_{chrg})^2 = 8 *(0.575)^2 = 2.64 \text{ [W]}$$

**CAUTION**:

The temperature that can reach a resistor working at is rated power can be as great as 90 degrees (Celsius). In general, the great the power rated, the great the volume of the resistor and thus, the lower the temperature rising. As rule of thumb, selecting a rated power twice the calculate power reduce the temperature rising to 40 degrees upon the ambient.

Mount this resistance in a place that natural or forced air convection ensures a good dissipation.

So a 5W resistor is selected.

The power required for the wall input adapter must include the charge current:

$$P_{wall} = P_{rasp}+P_{DV0}+P_{control}+P_{diode}+P_{charge}$$

*Where* $P_{charge}$ *is*

$$P_{charge} = V_{wall}\,(V_{wall} - V_{bat} - 0.2\,) / R_{chrg} = \\ = 12\,(12-7.2-0.2\,) /8 = 6.89 \text{ [W]}$$

And the rest of terms have been calculate above, so

$$P_{wall} = 6.26 + 1.675 + 0.3 + 0.17 + 6.89 = 15.3 \text{ [W]}$$

$$I_{Wall} = P_{Wall}/V_{Wall} = 15.3 / 12 = 1.275 \text{ [A]}$$

Figure 12 illustrates the schematic of wiring

**SUMMARY OF REQUIREMENTS:**

- Wall Input adapter: 12V, power > 15.3W, current > 1.275A
- Battery: NiMH, 7.2V 4.6 Amperes hour
- External charge resistor: 8Ω 5W

**Software:**

Example of *dv0_manager* call:

```
dv0_manager –force_date
```

Note that no minimum voltage is set because the default set of min_value is 6V, witch suits perfectly to a 7.2 NiMH battery

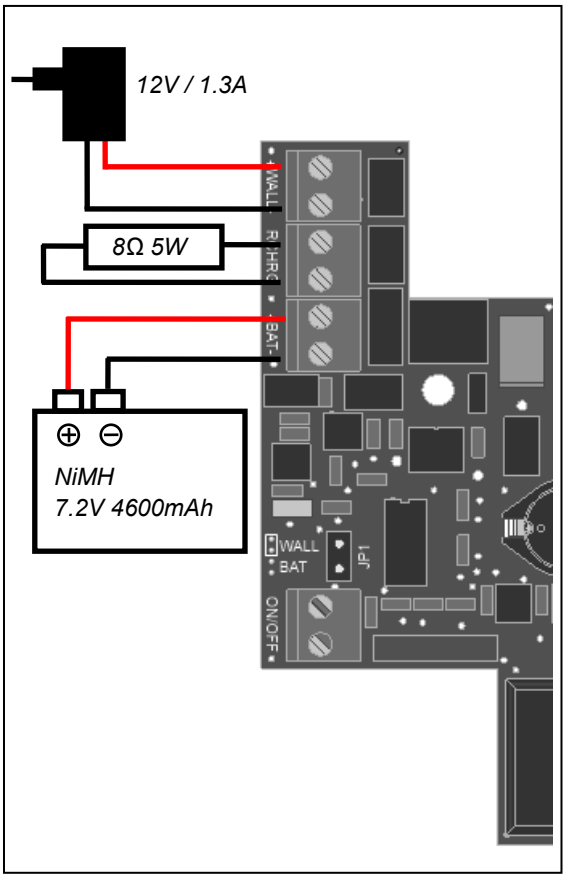



**Fig. 13 Battery mode example Case 2**

If eventually, the battery falls below 6V, the Header1 control unit will cleanly shutdown the Raspberry but even at 6V, there is some remained energy inside the battery and the control unit can maintains the real time clock for a while, until the wall input returns. Then, thanks to the *–force_date* parameter, the Linux operating system will restart with a correct time and date.

## 4. SOFTWARE

The Header1 board comes with several free programs, API and examples of C, Python, Java and Linux scripts. Their goal is to configure the power up/down behavior and scheduling and to allow user programs to gain access of the real word reading and writing DV0 input/output boards.

These are all the free software parts related to Header1:

- The ***dv0_manager*** program: the one and only one who talks with the Header1 control unit.
- The ***dv0_test*** utility: command program to test the *dv0_manager* and to send commands and settings to the Header1 control board. Intended to be used in Linux scripts
- The ***DV0 API libraries*** in C, Java and Python. A full set of functions to control the Header1 and the DV0 input/output board connected to the DV0 bus
- **Source code** examples written in C, Python and Java (Android)

All these parts and their associate documentation are free available in the Technical Information section of www.devalirian.com

### 4.1 Architecture description

Local architecture is the most common scenario and it is shown in the Figure 14.

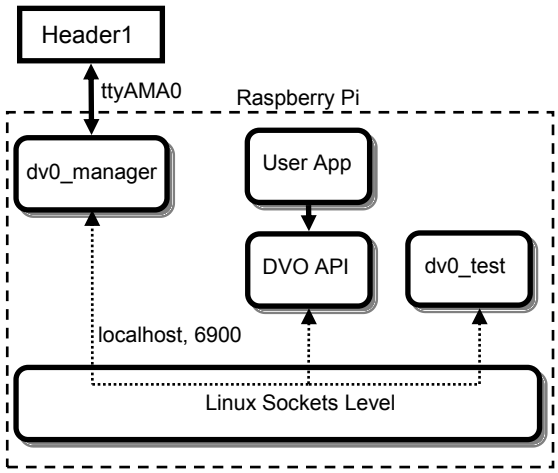



**Fig. 14 Local architecture**

In this case, the *dv0_manager* program waits for a socket connection listening to the interface "*localhost*" at port "6900". Both *dv0_test* program and user applications connect to *dv0_manager* using the function "*Open*", with that interface and port (which are the default values). The *dv0_manager* "tunnels" commands from *dv0_test* and user applications to the Header1 control unit through the Raspberry Pi serial interface *ttyAMA0*. Note that all elements run inside the Raspberry Pi.

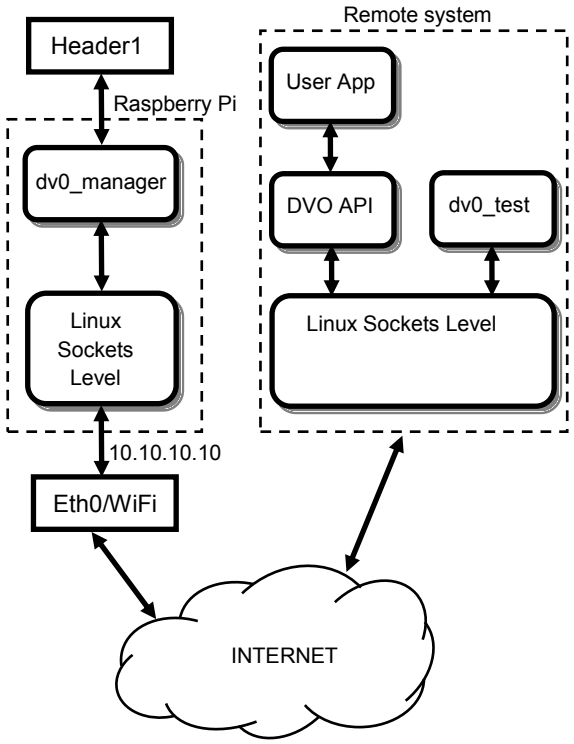



**Fig. 15 Remote access architecture**

Since the socket level physical layer is transparent to the user applications, it is possible to replicate the local architecture to a remote architecture, where the command and data traffic between *dv0_manager* and *DV0 API* or *dv0_test* travel through internet. Figure 15 shows this case.

In this example, the Raspberry is connected to internet via an interface which IP address is 10.10.10.10, no matter if it is an Ethernet or a WiFi connection. To accept data and commands from this interface as their come from localhost, only a few initialization changes must be performed. More exactly, the *dv0_manager* must be called with the –*ip_address* command line parameter :

```
dv0_manager -ip_address 10.10.10.10
```

and the "Open" function of DVO API requires a string with that IP address:

```
Open("10.10.10.10")
```

Note that when using a remote connection, some security issues must be taken into account to avoid unexpected and unwanted connection. See 4.3 for more information on login restrictions for a remote access.

## 4.2 *dv0_manager* installation

Log in using "pi" login name, download *dv0_manager* and *dv0_test* from the Technical Information section of www.devalirian.com and place into some directory. For instance, create and use /home/pi/dv0

Allow them execution permissions with *chmod* command:

```
chmod +x *
```

Edit the file */etc/rc.local* (you must need root permissions so remember to use the command *sudo* when invoking your preferred Linux editor. For example:

```
sudo nano /etc/rc.local
```

In almost all Linux distributions, this file is intended to contents the last commands that must be launched when the operating system boot is finish.

Add a line with the *dv0_manager* and all desired command line parameters and FINISH THE LINE with the ampersand character.

```
/home/pi/dv0/dv0_manager –force_date &
```

The inclusion of the ampersand character at the end of the line is mandatory or the Raspberry boot process never end. If you plan to launch *dv0_manager* in a separately script, it could be interesting to use its return value

### 4.2.1 *dv0_manager* return values

If there is something wrong, this program returns prematurely with the following codes:

- 255 (-1) if there is some syntax error at the command line
- 254 (-2) if the serial channel *ttyAMA0* is not free (see below)
- 253 (-3) if the socket library doesn't open the port 6900 or whatever declared with *–port* parameter. Try another port

### 4.2.2 Unblocking *ttyAMA0*

Since *dv0_manager* uses the serial channel *ttyAMA0* to communicate with the Header1 control board, this serial channel must be freely available, no other operating system part can share this channel with *dv0_manager*. Unfortunately, as a reminder of ancient times where the console was connected to a serial channel, the Raspberry Pi distros initial configuration take control of this channel as a root console.

To free this channel, edit the *boot/cmdline.txt* file with root permissions:

```
sudo nano /boot/cmdline.txt
```

and remove, carefully, both commands

```
console=ttyAMA0,115200
kgdboc=ttyAMA0,115200
```

Then, edit *inittab* file:

```
sudo nano /etc/inittab
```

and comment or remove lines that contain *ttyAMA0,* Finally, restart Linux and execute the *dv0_test* program:

```
cd /home/pi/dv0
./dv0_test
```

If every is fine, the program shows the message "Connection to localhost successful". Else, either there is a problem with the command line parameters, or with the serial channel or with the socket port.

To know what the problem is, begin with killing the running *dv0_manager* (if it still runs):

```
sudo killall dv0_manager
```

Then execute *dv0_manager* direct from the shell who will tell which one of the three failure condition explained in 4.2.1 is responsible for the malfunction.

## 4.3 *dv0_manager* command line parameters

This section describes the command line parameters accepted by the dv0_manager, their meaning, limits and default values. The "default value" is the value that *dv0_manager* uses if the corresponding parameter is not present at the command line call.

**Table 3. *dv0_manager* parameters summary**

| Parameter | Value | Default value |
|---|---|---|
| -serial_port | Port device | /dev/ttyAMA0 |
| -port | Port | 6900 |
| -ip_address | IP Address | localhost |
| -login | Login name | none |
| -shutdown_command | Command | sudo shutdown –h now |
| -shutdown_time | Seconds | 30 |
| -boot_time | Seconds | 30 |
| -restart_time | Minutes | 0 |
| -min_voltage | Tenths of V | 60 |
| -cut_vbus | None | false |
| -no_push_active | None | true |
| -force_date | None | false |
| -watch_dog | Seconds | 0 |

### 4.3.1 Parameter *-serial_port <port>*

Port is the name of the device where the header board is attached. If the header board is the Header1, this device is */dev/ttyAMA0*, but for others headers can be an USB-Serial emulator like */dev/ttyUSB0*.

**Default value**: */dev/ttyAMA0 (Raspberry Pi)*

### 4.3.2 Parameter *-port <port number>*

This is the TCP port number that dv0_manager listens to accept connections from *DV0 API* or *dv0_test*. It is not usual to change the default value of 6900 but in some cases, this value could be caught by other application.

**Default value:** 6900
**Minimum value:** 5000

### 4.3.3 Parameter *-ip_address <ip_address>*

This parameter is useful **only** when the user applications or the *dv0_test* run in other machine that the Raspberry Pi. The <*ipaddress*> must be equal to the interface IP where the connection to a local net is made. If all they run inside the same machine, the IP address is "localhost" which is the default value and so this parameter is not needed.

To know what IP address is assigned to an net port, execute the command:

```
ifconfig -a
```

For example, the result of this command can be:

```
eth0  Link encap:EthernetHWaddr b8:27:eb:c0:fb:76
      inet addr:10.0.0.2 Bcast:10.255.255.255
```

(…only two lines are shown…)

Then the call to *dv0_manager* should be

```
dv0_manager –ip_address 10.0.0.2
```

**Default value:** localhost

### 4.3.4 Parameter *-login <login name>*

When this parameter is set, all incoming connections to *dv0_manager* must be opened with the giving login name and with their corresponding password. For example, if the call to *dv0_manage*r is

```
dv0_manager –login pi
```

then, the initial DV0 API function Open must be

```
Open("localhost", 0, "pi", "raspberry")
```

and the *dv0_test* should be

```
dv0_test –login pi –password raspberry
```

Note that *dv0_manager* needs to gain access to the Linux password files in order to check the validity of the password and that means that it must be runnning with root privileges. See 4.6.1 for more information

### 4.3.5 Parameter *-shutdown_command "<command>"*

This is the command that the *dv0_manager* executes when a power off condition becomes true. Can be null, that is " " if no shutdown generation is desired.

**Default value:** *"sudo shutdown –h now"*

### 4.3.6 Parameter *-shutdown_time <seconds>*

Number of seconds that the Header1 control unit waits before cutting off the power to the Raspberry Pi once there is a Power Off condition true

**Default value:** 60
**Maximum value:** 32767

### 4.3.7 Parameter *-boot_time <seconds>*

Number of seconds that the Header1 control unit waits before considering that the boot process is finished and thus accept and check Power Off conditions.

**Default value:** 60
**Maximum value:** 32767

### 4.3.8 Parameter *-restart_time <minutes>*

This parameter defines the number of minutes that the Header1 control unit must wait to generate a Power On condition after a power off condition has been execute successfully. See 1.1.2 Power Off Conditions for more explanations about their causes.

Intended to generate a full hardware reset after a watch dog condition or when a complete system initialization is required. See 4.5 for useful Linux scripts examples using this parameter and the *dv0_manager*.

Notes:

- This timer begins to run AFTER the time defined by the *shutdown_time* parameter.
- A value of zero means that the Raspberry never will be restarted.
- Restarting is only allowed if the input voltage is greater than 6V whichever Battery or Wall input is the greater one.
- Once restarted, the Header1 control units forgets this value so dv0_manager must be called with this parameter again.

**Default value:** 0 (Never)
**Maximum value:** 32767

### 4.3.9 Parameter *-min_voltage* <voltage>

Defines the threshold voltage (in tenths of volts) of the battery voltage that generates a power off or power on condition (if it last for a 5 seconds). The power off level is exactly this value, while the power on condition is at 200mV over this value. This *hysteresis* behavior avoids noise dependence when the battery voltages is close to *min_voltage*. For example, if the desired threshold is 11.9V, then

```
dv0_manager –min_voltage 119
```

In this example, power off condition is at 11.9, but power on waits until the voltage exceeds 12.1V.

**Default value:** 60 (6V)
**Minimum value:** 60 (6V)
**Maximum value:** 240 (240V)

### 4.3.10 Parameter *-cut_vbus*

This parameter forces the Header1 control unit to cut the 5V power of DV0 bus as soon as a Power Off condition becomes true, without waiting for the *shutdown_time*.

Very useful in wall mode when a little battery is used to assure a clean shutdown. However, there are situations when the user application must set all the input/output signals to a pre-defined state before power off and then this parameter can't be set.

**Default value**: false (DV0 voltage falls after the shutdown time, together with the voltage of the Raspberry board)

### 4.3.11 Parameter *-no_push_active*

Setting this parameter forces the Header1 control unit to ignore the push button input signal

**Default value**: false (Header1 control unit doesn't ignore the push button signal)

### 4.3.12 Parameter *-force_date*

When this parameter is set, *dv0_manager* sets the Linux date and time kept by the Header1 control unit every time that the system restart.

Header1 control unit date and time can be set using either the *dv0_program* or the API function *SetDate* (see 4.6.2)

**Default value**: false (date is not set)

### 4.3.13 Parameter *-start_up_time dd:hh:mm*

Defines the periodicity at which the Header1 control unit will power up the system in days, hours and minutes.

Examples:

*-start_time 10* Means every hour at minute 10
*-start_time 8:10* Means every day at 8 hours and 10 minutes
*-start_time 3:08:10* Means every month, at day 3, at day at 8 hours and 10 minutes

Note: the value's correctness is not check. If there is something impossible about days, hours or minutes, *dv0_manager* will assume as 1.

**Default value**: never

### 4.3.14 Parameter *-watch_dog <seconds>*

If this parameter is set and greater than zero, the user must issue a watchdog reset periodically with a period smaller than the number of seconds specified. If not, dv0_manager will generate a Power Off sequence. Even if the *dv0_manager* is hanged and don't issue a proper shutdown command, the Header1 control unit will power off the Raspberry Pi (to restart automatically after one minute, set *-restart_time 1*).

This feature is specially for non assisted systems to restart the Raspberry if some software dysfunction or

electromagnetic interference hangs the Linux of the Raspberry.

**Default value**: 0 (No watch dog reset)
**Maximum value:** 255

## 4.4 *dv0_test* command line parameters

The dv0_test program is intended to be a system tool for testing board connections, verifying *dv0_manager* correctness and programming Linux scripts.

This section describes the command line parameters accepted by the dv0_test, their meaning, limits and default values. Table 4 summarizes these parameters

**Table 4. *dv0_manager* parameters summary**

| Parameter | Value | Default value |
|---|---|---|
| -ip_address | IP Address | localhost |
| -port | Port number | 6900 |
| -login | Login name | None |
| -password | Password | None |
| -set_date | None | None |
| -set_restart_time | Minutes | 0 |
| -power_off | None | None |
| -get_manifest | Address | None |
| -watch_dog_reset | None | None |

### 4.4.1 Parameter *-ip_address <ip address>*

Use this parameter when *dv0_manager* has been called with this parameter as well. As explained in 4.3.3 section, if *dv0_manager* is invoked as

```
dv0_manager –ip_address 10.0.0.2
```

then, to communicate with *dv0_manager*, *dv0_test* must be called as

```
dv0_test –ip_address 10.0.0.2
```

NOTE: when both *dv0_manager* and *dv0_test* run in the same machine, they can use the interface “localhost” to establish the link, which is the default value for both of them. In this case, there is no need to set this parameter.

### 4.4.2 Parameter *-port <Port number>*

This parameter must be set only if the default port 6900 is occupied by another application forcing dv0_manager to be called with this parameter. For example:

```
dv0_manager –port 11200
```

then,

```
dv0_test –port 11200
```

### 4.4.3 Parameter *-login <login name>*

When *dv0_manager* has been called with this parameter then all further connections to its must supply not only the same login name but its password.

For example, as in the 4.3.4 section, when *dv0_manager* is called like

```
dv0_manager –login pi
```

then, dv0_test must supply the same login and its valid password

```
dv0_test –login pi –password raspberry
```

Note: This password is not encrypted while travelling through internet.

### 4.4.4 Parameter *-password <password>*

Used always together with *-login* parameter described above.

### 4.4.5 Parameter *-set_date*

This parameters forces *dv0_manager* to store the Linux date and time into the Header1 control unit. The real time clock of this control unit will continue running from this value and used for the *-start_up_time* parameter and to restore it to the Linux at Power On time (only if parameter *-force_date* has been set for the *dv0_manager* program),

### 4.4.6 Parameter *-set_restart_time <minutes>*

Tells dv0_manager the number of minutes that will take the next restart time. It overrides the *dv0_manager -restart_time* parameter .

### 4.4.7 Parameter *-power_off*

This parameter forces *dv0_manager* to start a Power Off process.

### 4.4.8 Parameter *-get_manifest <address>*

The manifest is a readable text who explains the features available from an DVO input/output board. The value of <address> must match with the micro-switch selector of the board that has to be tested.

For example, if there is a model IO_A board connected which micro-switch address selector is 7, then, when executing

```
dv0_test –get_manifest 7
```

the response will be

```
Board name    : IO_A
PWM Outputs   : 3
Pulse Inputs  : 3
Analog Inputs : 2
Digital output group number 0 has 2 outputs
Digital output group number 1 has 4 outputs
Digital input group number 0 has 3 inputs
```

Which are the contents of a model IO_A board.

It is worth to note that if this command is successful, that means that the whole installation is correct: *dvo_manager* is running, it is connected with the Header1 control unit and the Header1 DV0 Bus cabling is correct as well.

### 4.4.9 Parameter *-watch_dog_reset*

When the *–watch_dog* parameter is set for more than 0 seconds at the dv0_manager command line, someone MUST send a watch dog reset command to the Header1 control unit. If not, the Raspberry Pi will be restarted.

Using dv0_test with *-watch_dog_resets* periodically is one way to avoid restarting (the other way is through the DVO API function *ResetWatchDog*)

See 4.5 for an example on how to reset the watch dog using *crontab* and *dv0_test*.

### 4.4.10 Return values of *dv0_test*

On successful, dv0_test returns 0. If not, it returns prematurely with the following return values

- 255 (-1) dv0_manager does not respond. Check IpAddress and Port
- 254 (-2) dv0_manager refuses login or password
- 253 (-3) DVO header board is not connected
- 252 (-5) Invalid board address
- 251 (-6) Parameter syntax error

## 4.5 Examples of dv0_manager, dv0_test and Linux scripts

This section illustrates some configuration of *dv0_manager* and the corresponding calls of dv0_test.

**Note**: In all following examples, it is assumed that the *dv0_manager* runs in the directory */home/pi/dv0* and that in the */etc/rc.local*, there is the command:

```
cd /home/pi/dv0
```

prior to call dv0_manager. For example, the simplest installation of dv0_manager is just edit */etc/rc.local* and write into it:

```
cd /home/pi/dv0
./dv0_manager &
```

Note that the final ampersand is always mandatory.

Figure 17 shows how to fit the *shutdown_time* and the *boot_time* (once they are measured) and how to force the date to the system at power up. Designers must find some way to obtain the actual date and time (at least, one time) and then, call *dv0_test* with *-set_date* parameter while Figure 18 shows the same scenario but in a remote architecture

Figure 19 shows how to power off and restart the whole system (Raspberry board and DVO boards). When issuing a *-power_off* without setting the restart time, then there is no automatic restart. Restart time can be set when calling *dv0_test* or when calling *dv0_manager*.

Figure 20 illustrates how to customize the shutdown command so that a previous settings of the DV0 boards outputs are accomplished before issuing the true shutdown.

File /etc/rc.local:

```
./dv0_manager -force_date -shutdown_time 20 -boot_time 30 &
```

Somewhere once the system date is well known:

```
./dv0_test -set_date
```

**Fig. 17 Local architecture with forcing date and fitted boot and shutdown time**

File /etc/rc.local:

```
./dv0_manager -ip_address 10.0.0.1 -force_date -shutdown_time 20 -boot_time 30 &
```

Somewhere once the system date is well known (the date comes from the machine where *dv0_manager* runs):

```
./dv0_test -ip_address 10.0.0.1 -set_date
```

**Fig. 18 Remote architecture with forcing date and fitted boot and shutdown time**

File /etc/rc.local:

```
./dv0_manager -force_date -shutdown_time 20 -boot_time 30 &
```

Just power off, and power on from the pushbutton or turning off-on input power:

```
./dv0_test -power_off
```

Forcing a restart one minute after the shutdown process finished:

```
./dv0_test -power_off -set_restart_time 1
```

Another way to force a power off: sending a SIGPWR to dv0_manager:

```
sudo killall -s SIGPWR dv0_manager
```

**Fig. 19 Several ways to power off and to restart with fitted boot and shutdown time**

To do that, dv0_manager is set to call a script named *my_script*, who, thanks to a user program based on the DV0 API, sets the outputs and then executes the true shutdown.

The new script “my_script” will be executed every time that a Power Off condition becomes true.

Finally, a script for managing the return value of *dv0_test* is shown in the Figure 22. This scripts begins calling *dv0_test* with the *-get_manifest* of the board number supplied as a first parameter of the script. Then, the variable *$Res* holds de result of *dv0_test* (“$&”) and compares it with all possible return values described in 4.4.10

File /etc/rc.local:

```
./dv0_manager -shutdown_command my_script &
```

Edit my_script and save it in the same directory of dv0_manager (/home/pi/dv0):

```
# do some DV0 API based program and then issue a Linux shutdown
sudo shutdown –h now
```

Don’t forget to make it executable:

```
sudo +x my_script
```

**Fig. 20 Customized shutdown**

File /etc/rc.local:

```
./dv0_manager –watch_dog 180 &
```

Edit /etc/crontab and add the following line

```
*/1 * * * * /home/pi/dv0/dv0_test -watch_dog_reset
```

**Fig. 21 Watchdog reset from *crontab* using *dv0_test***

```
./dv0_test -get_manifest $1 1>&null
Res="$?"
if [ "$Res" = "0" ]; then
        echo "Board is Ok"
elif [ "$Res" = "255" ]; then
        echo "dv0_manager does not respond"
elif [ "$Res" = "254" ]; then
        echo "invalid login or password"
elif  [ "$Res" = "253" ]; then
        echo "Header not connected"
elif  [ "$Res" = "252" ]; then
        echo "Invalid board address"
else
    echo "Parameter syntax error"
fi
```

**Fig. 22 Script for managing dv0_test return values**

Figure 21 describes how to protect the system by a watch dog. The dv0_manager is call with the parameter *-watch_dog* set to 180 seconds, so someone must reset periodically this timer by calling *dv0_test* with the *-wath_dog_reset*. The obvious solution is to add a line in the */etc/crontab* that call it every minute.

## 4.6 *DV0* API

The DV0 API contains a set of functions intended to manage the Header1 control unit and all DVO input/output expansion board connected to the bus. There are a free implementation of this API, for Java, C++ and Python languages.

**WARNING:**

This data sheet summarizes the features of DV0 API functions related to Header1 board. It is not intended to be a comprehensive reference source. For more information, refer to the DV0 API Reference Guide for C++, Java and Python at www.devalirian.com, in the Technical Information section

The steps for managing the Header1 control board programmatically are:

- Create an instance of the class DV0
- Call the function Open
- If successful, call Header1 functions related.

In the following sections, an example of each language will be given creating instance, calling Open function and calling related functions and more complete examples can be found at the end of this section.

Table 5 summarize the functions related to Header1. Note that only the name of the function is listed in the table below. This is because the parameters and return values are slightly different among the three implementation (Java, C++ and Python).

**Table 5. Header1 related functions**

| Name | Description |
|---|---|
| **Open** | Initial and mandatory function to connect with dv0_manager |
| **SetDate** | Set the date and time of Header1 control unit |
| **GetDate** | Get the current date and time kept by the Header1 control unit |
| **SetRestartTime** | Set restart time |
| **PowerOff** | Starts a power off sequence |
| **ResetWatchDog** | Clears the watch dog timer (if enabled) |
| **GetHeaderInfo** | Obtains status and version info of the Header1 control unit |

### 4.6.1 Creating an instance of DV0 class

Java, C++ and Python are all object oriented languages and is useful to encapsulate all the accessing to the API throughout one object. All subsequent managing functions will pass through this variable. Let's assume that this variable will be called *dv0*. For C++ or Java, the creation of this variable is thoroughly:

```
DV0 dv0; // C++ or Java
```

And so is for Python, but slightly different (actually, it is not a class)

```
import dv0
```

### 4.6.2 Connecting to *dv0_manager*

The connection to *dv0_manager* is made through the function Open.

| Function *Open* generic description |
|---|
| `int Open(IpAddress, Port, Login, Password);` |
| Open connection with the dv0_manager. Mandatory for all other members. Blocking. |
| **Parameters:** |
| **IpAddress** is the IP where dv0_manager listens to incoming connections. By default is "localhost" that suits perfectly is this application runs in the same machine that dv0_manager does. If a remote connection is desired, call dv0_manager with the argument -ip_address followed by the IP address of the internet interface of your board (use ifconfig to know it) and pass this value to the current IpAddress parameter.<br>**Port** is the UDP port where dv0_manager listens to incoming connections. By default is 6900 but DV0_manager can listen to whatever port selected by command line argument -port. In this case, give the same value to current Port parameter. Useful only if another application had catch this port.<br>**Login** and **Password** are required if the dv0_manager daemon is called with -login <username> argument. Current Login parameter must match with <username> and a valid Password must be entered |
| **Return values** |
| Returns 0 if connection is successful<br>DV_NOT_CONNECTION if dv0_manager does not respond. Check IpAddress and Port<br>DV_INVALID_LOGIN if dv0_manager refuses login or password |
| **C++ Syntax** |
| `int Open(char *IpAddress = NULL, int Port = 0, char *Login = NULL, char *Password = NULL);` |
| **Java Syntax** |
| `int Open(String IpAddress, int Port, String Login, String Password);` |
| **Python Syntax** |
| `def Open(ipAddress, port, login, password):` |

**Example**: Connect to a *dv0_manager* running in the same Raspberry that the current program, using de default port and with no login required. The *dv0_manager* call can be:

```
/home/pi/dv0/dv0_manager –force_date
```

From a C++ user program, then:

```
DV0 dv0;
…
int r = dv0.Open(NULL, 0, NULL, NULL)
if (r == 0) {
…
} else DisplayError(r);
```

From a Java user program, then:

```
DV0 dv0;
…
int r = dv0.Open(””, 0, ””, ””)
if (r == 0) {
…
} else DisplayError(r);
```

From a Python program, there are no so much differences:

```
import dv0
res = dv0.Open('',0,'','')
if (res == 0):
    print ("Connected to dv0_manager")
else:
```

```
    DisplayError(res)
```

**Example**: Connect to a *dv0_manager* running in a Raspberry connected to internet through the interface 10.0.0.2, and the current program running somewhere, login name 'pi' required (with default password 'raspberry') , using de default port. The *dv0_manager* call can be:

```
/home/pi/dv0/dv0_manager -ip_address 10.0.0.2 -force_date -login pi
```

From a C++ and Java user program, then

```
DV0 dv0;
…
int r = dv0.Open("10.0.0.2", 0, "pi", "raspberry")
if (r == 0) {
…
} else DisplayError(r);
```

From a Python program, there are no so much differences:

```
import dv0
res = dv0.Open('10.0.0.2',0,'pi','raspberry')
if (res == 0):
    print ("Connected to dv0_manager")
else:
    DisplayError(res)
```

### 4.6.3 API functions related to Header1

Functions *SetDate* and *GateDate* manage the real time clock of Header1 control unit

| Function *SetDate* generic description |
|---|
| `int SetDate(int Year, int Month, int Day, int Hour, int Minute, int Second)` |
| Sets this date to the real time clock embedded in the Header1 board |
| **Parameters:** |
| Pre-condition : User must pass a valid range for each parameter and a valid global date.<br>Unpredicted behavior can occurs if an invalid date is entered.<br>Valid values: Year = 2000..2099; Month = 1..12; Day = 1..31; Hour = 0..23;<br>Minute = 0..59; Second = 0..59 |
| **Return values** |
| Returns 0 if successful<br>DV_NOT_CONNECTED if previous Open call had failed or connection has been canceled |
| **C++ Syntax** |
| `int SetDate(int Year, int Month, int Day, int Hour, int Minute, int Second);` |
| **Java Syntax** |
| `int SetDate(int Year, int Month, int Day, int Hour, int Minute, int Second);` |
| **Python Syntax** |
| `def SetDate(Year, Month, Day, Hour, Minute, Second):` |

Function *GetDate* generic description

```
int GetDate(int Year, int Month, int Day, int Hour, int Minute, int Second)
```

Fills parameters with the current real time clock kept by the Header1 board

Parameters:

Return values

| | |
|---|---|
| Returns 0 | if successful |
| DV_NOT_CONNECTED | if previous Open call had failed or connection has been canceled |

C++ Syntax

```
int GetDate(int &Year, int &Month, int &Day, int &Hour, int &Minute, int &Second);
```

Java Syntax

```
int GetDate(int Date[]);
whereas Date[0] = Year, Date[1] = Month, Date[2] = Day, Date[3] = Hour, Date[4] = Minute
and Date[5] = Second;
```

Python Syntax

```
def GetDate():
# Returns a 2-elements tuple containing an array Date (as in Java) and an integer
# (return value)
    return Date, Result
```

Function *SetRestartTime* generic description

```
int SetRestartTime(int Minutes)
```

Informs the Header1 board that the current restart time is "minutes", which overloads the argument -restart_time given at the invocation of dv0_manager

Parameters:

Number of minutes

Return values

| | |
|---|---|
| Returns 0 | if successful |
| DV_NOT_CONNECTED | if previous Open call had failed or connection has been canceled |

C++ Syntax

```
int SetRestartTime(int Minutes);
```

Java Syntax

```
int SetRestartTime(int Minutes);
```

Python Syntax

```
def SetRestartTime(Minutes):
```

Function *PowerOff* description

```
int PowerOff()
```

Tells the Header1 board to initiate a shutdown process as if it was initiated by the pushbutton or power failure

Parameters:

Return values

| | |
|---|---|
| Returns 0 | if successful |
| DV_NOT_CONNECTED | if previous Open call had failed or connection has been canceled |

C++ Syntax

```
int PowerOff ();
```

Java Syntax

```
int PowerOff ();
```

Python Syntax

```
def PowerOff ():
```

Function *ResetWatchDog* description

```
int ResetWatchDog ()
```

Tells the Header1 board to reset the watchdog timer, if it was activated by the parameter -watch_dog <seconds> of dv0_manager.

Parameters:

Return values

| | |
|---|---|
| Returns 0 | if successful |
| DV_NOT_CONNECTED | if previous Open call had failed or connection has been canceled |

C++ Syntax

```
int ResetWatchDog ();
```

Java Syntax

```
int ResetWatchDog ();
```

Python Syntax

```
def ResetWatchDog ():
```

Function *GetHeaderInfo* description

```
int GetHeaderInfo (Info[8])
```

Fills the array Info with information from the Header1 control unit.

Parameters:

The meaning of each occurrence of the array Info is
Info[0] = Header1 version number
Info[1] = Voltage at Wall input (in tenths of volts)
Info[2] = Voltage at Battery input (in tenths of volts)
Info[3] = Number of DV0 transmission errors detected
Info[4] = Number of DV0 Input/Output boards that have been detected
Info[5] = Percent of Header1 control unit's memory used
Info[6] = Unused
Info[7] = Unused

Return values

| | |
|---|---|
| Returns 0 | if successful |
| DV_NOT_CONNECTED | if previous Open call had failed or connection has been canceled |

C++ Syntax

```
int GetHeaderInfo (unsigned char Info[8]);
```

Java Syntax

```
int GetHeaderInfo (byte Info[8]);
```

Python Syntax

```
def GetHeaderInfo ():
# Returns a 2-elements tuple containing an array Info (as in C++ or java) and an integer
# (return value)
     return Date, Result
```

### 4.6.4 API examples

In this section, a complete example of how to manage Header1 related functions are described both in C++ and Python languages. Java example is omitted because it is so close to C++ that only adds confusion. However, there is an example of how to program an Android APP with the DV0 API, that is not included in this section for the benefit of simplicity but it can be found at www.devalirian.com, inside the Technical Information section.

#### 4.6.4.1. C++ ExampleHeader1.cpp

```
#include <iostream>
using namespace std;
#include "DV0.h"

char *ServAddress = NULL; char *LoginName = NULL; char *Password = NULL; int Port = 0;
DV0 dv0;

void DisplayError(int r);
int DoTest(void);
int SetDate();

int main() {
	int r;
	// Trying to connect
	r = dv0.Open(ServAddress, Port, LoginName, Password);
	if (r == 0) {
		cout << "Connection to dv0_manager successful " << endl;
		r = DoTest();
		if (r != 0) DisplayError(r);
	} else DisplayError(r);
	return r;
}

int DoTest(void) {
	int option, r, y, m, d, h, min, sec;
	// Print menu
	cout << "Menu:" << endl;
	cout << "1-SetDate 2-GetDate 3-SetRestartTime 4-PowerOff" << endl;
	cout << "5-ResetWatchDog 6-GetHeaderInfo" << endl;
	// Get user option
	cin >> option;
	// And execute
	switch (option) {
		case 1:
			//----------------------------------------Set Date
			return SetDate();
		case 2:
			//----------------------------------------Get Date
			r = dv0.GetDate(y,m,d,h,min,sec);
			if (r == 0) {
				cout	<< "Date is "<< y <<"/"<< m <<"/"<< d
					<<" "<< h <<":"<< min <<":"<< sec << endl;
```

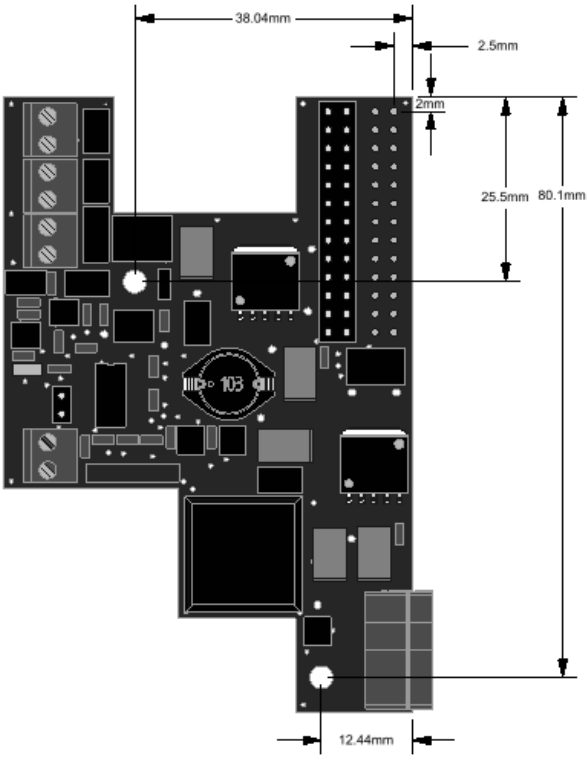
```
                        }
                        return r;
                case 3:
                        //---------------------------------------SetRestartTime
                        cout << "Restart time (minutes):" << endl; cin >> min;
                        return dv0.SetRestartTime(min);
                case 4:
                        //---------------------------------------PowerOff
                        return dv0.PowerOff();
                case 5:
                        //---------------------------------------ResetWatchDog
                        return dv0.ResetWatchDog();
                case 6:
                        //---------------------------------------GetHeaderInfo
                        unsigned char Info[8];
                        r = dv0.GetHeaderInfo(Info);
                        if (r == 0) {
                                cout << "Version: " << (int)Info[0] << endl;
                                cout << "VIn    : " << (float)Info[1]/10.0 << endl;
                                cout << "VBat   : " << (float)Info[2]/10.0 << endl;
                                cout << "CRC err: " << (int)Info[3] << endl;
                                cout << "Boards : " << (int)Info[4] << endl;
                                cout << "Mem %  : " << (int)Info[5] << endl;
                        }
                        return r;
        }
        cout << "Invalid option" << endl;
        return 0;
}

int SetDate() {
// This function gets the current system time and sends it to the Header1 control unit
        time_t t = time(NULL);
        struct tm tm = *localtime(&t);
        return dv0.SetDate(tm.tm_year+1900,tm.tm_mon+1,
                        tm.tm_mday,tm.tm_hour,tm.tm_min,tm.tm_sec);
}

void DisplayError(int error) {
        switch (error) {
                case DV_NOT_CONNECTION:
                        cout << "dv0_manager does not respond. Check IpAddress and Port" << endl;
                        break;
                case DV_INVALID_LOGIN:
                        cout << "dv0_manager refuses login or password" << endl;
                        break;
                case DV_NOT_CONNECTED:
                        cout << "DVO header board is not connected" << endl;
                        break;
                case DV_INVALID_BOARD:
                        cout << "Invalid board address" << endl;
                        break;
                case DV_NOT_SUPPORTED:
                        cout << "Invalid peripheral" << endl;
                        break;
                default:
                        cout << "Unexpected error code" << endl;
                        break;
        }
}
```

#### 4.6.4.2. Python ExampleHeader1.py

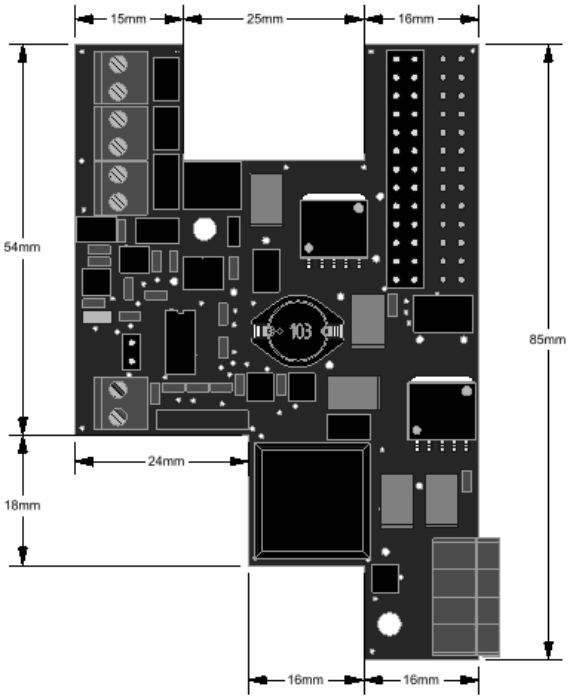
```
import sys
import dv0

def DisplayError(error):
    if error == 0:
        return
    if error == dv0.DV_NOT_CONNECTION:
        print ("dv0_manager does not respond. Check IpAddress and Port")
    elif error == dv0.DV_INVALID_LOGIN:
        print ("dv0_manager refuses login or password")
    elif error == dv0.DV_NOT_CONNECTED:
        print ("DVO header board is not connected")
    elif error == dv0.DV_INVALID_BOARD:
        print ("Invalid board address")
    elif error == dv0.DV_NOT_SUPPORTED:
        print ("Invalid peripheral")
    else:
        print ("Unexpected error code: ", error)

# Assuming that the Raspberry is not this computer and connected as 10.0.0.2
# Also, dv0_manager is called like "dv0_manager -ip_address 10.0.0.2 -login pi"
res = dv0.Open('10.0.0.2', 0, 'pi', 'raspberry')
# Assuming that the Raspberry is this computer
# and dv0_manager is called just like "dv0_manager"
# res = dv0.Open('',0,'','');

looping = True
if (res == 0):
    print ("Connected to dv0_manager")
else:
    DisplayError(res)
    looping = False
while looping:
    print ("1-SetDate 2-GetDate 3-SetRestartTime 4-PowerOff")
    print ("5-ResetWatchDog 6-GetHeaderInfo")
    option = int(input("Enter option:"))
    if option == 1:
        # ----------------------------------------- Set Date example
        year   = int(input ("Enter year :"))
        month  = int(input ("Enter month:"))
        day    = int(input ("Enter day  :"))
        hour   = int(input ("Enter hour :"))
        minute = int(input ("Enter minute :"))
        second = int(input ("Enter second :"))
        res = dv0.SetDate(year, month, day, hour, minute, second)
        DisplayError(res)
    elif option == 2:
        # ----------------------------------------- Get Date example
        Date,res = dv0.GetDate()
        if res == 0:
            print("Year:", Date[0], " Month:", Date[1], " Day:", Date[2],
              " Hour:", Date[3], " Minute:", Date[4], "Second: ", Date[5])
        else:
            DisplayError(res)
    elif option == 3:
        # ----------------------------------------- Set restart time example
```

```
        minutes = int(input ("Enter restart time in minutes:"))
        res = dv0.SetRestartTime(minutes)
        DisplayError(res)
    elif option == 4:
        # ---------------------------------------- Power Off example
        res = dv0.PowerOff()
        DisplayError(res)
    elif option == 5:
        # ---------------------------------------- Reset Watch Dog example
        res = dv0.ResetWatchDog()
        DisplayError(res)
    elif option == 6:
        # ---------------------------------------- Get Header1 or HeaderUSB example
        Info, res = dv0.GetHeaderInfo()
        if (res == dv0.DV0_OK):
            print ("Version: %d", Info[0]);
            print ("VIn    : %d", Info[1]/10);
            print ("VBat   : %d", Info[2]/10);
            print ("CRC err: %d", Info[3]);
            print ("Boards : %d", Info[4]);
            print ("Mem %  : %d", Info[5]);
        else:
            DisplayError(res)
```

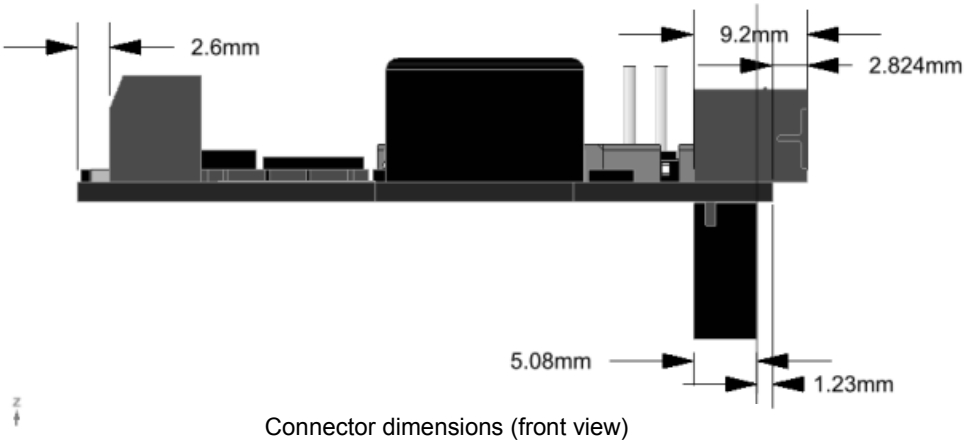
## 5. MECHANICAL DRAWINGS



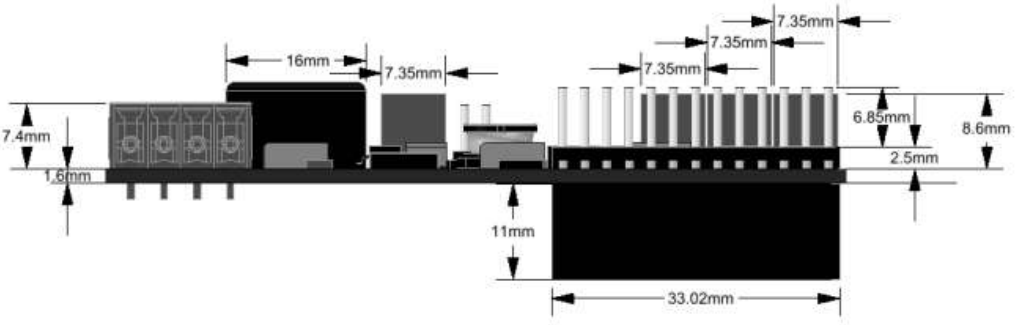

Hole dimensions (top view)




Board dimensions (top view)




Connector dimensions (front view)



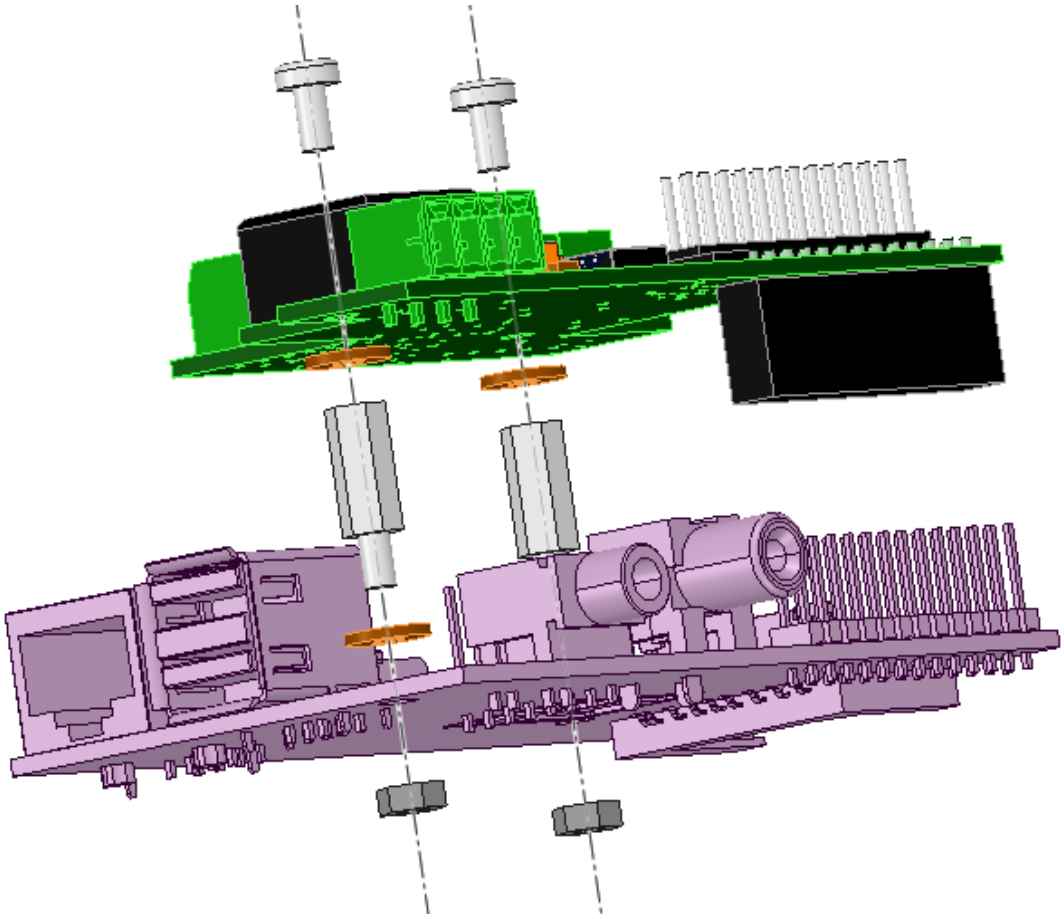

Connector dimensions (left view)

## 6. ACCESSORIES

The Header1 control board comes with the following accessories:

- One DV0 socket. Model 20020004-D041B01LF
- Two Spaces (M3, 12mm). Model SP1112
- Four bakelite washers. Model AB14
- Two M3 x 6mm screw. Model M3X6EE
- Two M3 nut. Model TRM3

The spacers, washers, nuts and screws are intended to assembly the Header1 control board to the Rapberry Pi board as shown in the following picture



## IMPORTANT NOTICE

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications.

***deValirian*** MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE**.**

***deValirian*** disclaims all liability arising from this information and its use. Use of ***deValirian*** devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless ***deValirian*** from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any ***deValirian*** intellectual property rights.

Header1 board is not designed to be radiation tolerant

Please be sure to implement in your equipment using the safety measures to guard against the possibility of physical injury, file or any other damaged cause in event of the failure of Header1 board. deValirian shall bear no responsibility whatsoever for you're your use of Header1 board outside the prescribed scope or not in accordance with this manual.

Reproduction of significant portions of ***deValirian*** information in ***deValirian*** data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. ***deValirian*** is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Web Site: www.devalirian.com

Mail info: info@devalirian.com