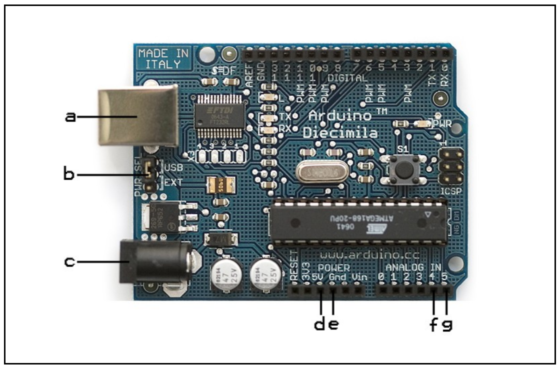
# RJSC01 Overview

## Technical Specifications

The RJSC01 (RoboJacket Sonar Controller) can control an array of up to sixteen SRF08 sonar devices via a standard I2C communication bus. Communication with the controller itself is through the USB port. The RJSC01 utilizes the Atmega168 microcontroller to carry out the many functionalities implemented within the system. For further details concerning the SRF08 or Atmega168 architectures, see their respective datasheets.

**Electrical Setup.** The RJSC01 is built into the Arduino Decimilia environment. This allows for simple access to the board's power input **(c)**, USB port **(a)**, and various I/O pins. The Decimilia may be powered by either an input power plug (7~12V) or through it's USB port. The jumper pins **(b)** for power source selection is located next to the USB connection port.



*Arduino Diecimila*

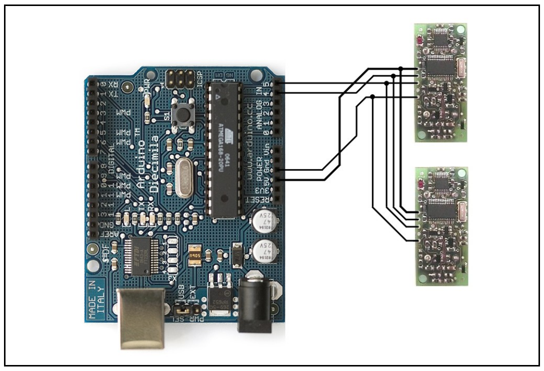


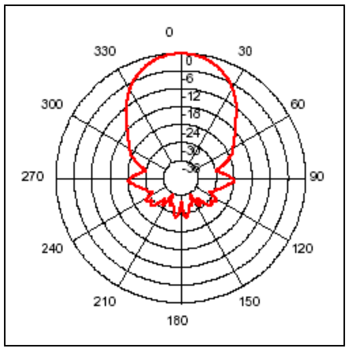
*Illustration 1: I2C Bus Connection*

Analog pins 4 **(f)** and 5 **(g)** control the SDA and SCL lines of the I2C bus, respectively. Power **(d)** and ground **(e)** lines can be drawn from the the Arduino board directly. All four lines are required for successful operation of any SRF08 device. See the image above for details on connecting the Arduino to an SRF08 array.

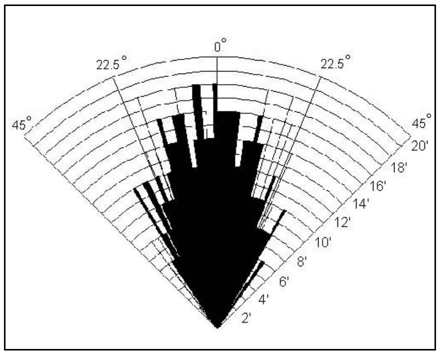
Electrical specifications are detailed in Table 1, below. Do note that, although several SRF08's may be pinged at the same time, this can cause noise in distance readings. A beam pattern from the manufacturer's datasheet is also displayed.

| Operation | Current | Duration |
|---|---|---|
| PWR on / Ranging Cmd | 275 mA | 3 µs |
| ±10V Generator Stab. | 25 mA | 600 µs |
| Pinging | 40 mA | 200 µs |
| Ranging | 11 mA | 65 ms (max) |
| Standby | 3 mA | -- |

*Table 1:Electrical Characteristics*



*SRF08 Beam Pattern*



*SRF08 Beam Test*

**Modes and Functions.** Many functions and various modes of operation have been implemented into the RJSC01 for efficient and flexible control over any given SRF08 sensor array arrangement. A brief outline has been provided below, however an exhaustive list of each function can be found in the **Functions and Compilation** segment of this article.

Setup Functions.

| | |
|---|---|
| **Initialize-** | Required for properly setting up the RJSC01 system. It is called at the beginning of the program and, as the name implies, initializes various components of the system. |
| **Set Step Total-** | Initializes the total number of steps that the system should allow for. Steps are used on this system so that the SRF's in an array can be pinged in a specified order. Each device is given a step ID, and can only ping on the step that corresponds with that number. A demo of this functionality can be found in the **Code Samples** section. |
| **Set Individual Step-** | Assigns a step identification number to the specified SRF. |
| **Set Frequency-** | Sets the frequency of the system in units of steps per second. Note: this does <u>not</u> change the operating frequency of the transducers. |
| **Set Device Active-** | Allows user to activate or deactivate the specified SRF. A deactivated device does not take part in pinging or reading. |

Device Functions.

| | |
|---|---|
| **Ping-** | Commands the specified SRF to ping. |
| **Read-** | Reads back the most recent value obtained from a SRF ping. This value is always in millimeters. |
| **Ping All-** | Commands the entire SRF array to ping in order. |

Parameter Functions.

| | |
|---|---|
| **Set Gain-** | Sets the gain of a specified SRF. Gain values can be between 0 and 31, which correspond to analog gain values between 94 and 1025. See the SRF08 datasheet for details. |
| **Set Max Range-** | Sets the maximum range that the SRF can detect. A smaller range value will correspond to a shorter time spent pinging. |
| **Set Autopilot On/Off-** | Allows user to activate or deactivate autopilot mode. In this mode, a 'ping all' function runs immediately after the latest range values are read from the RJSC01. |

**Functions and Compilation.** The software interface for the RJSC01 is fairly simple and intuitive. Several libraries are required for interface between the computer and the system: SonarInterface, ArduinoInterface, and DataPacket. SonarInterface deals specifically with RJSC01 related functions. This is the only library that the user should be calling functions from. The other two libraries are used implicitely within SonarInterface. ArduinoInterface provides general communication between the computer and the Atmega168. DataPacket is used for packetization and data handling. Several other source files are also required. A full list of files needed for successful compilation are listed below and can be found on the Robojacket's IGVC 2009 SVN under the Sonar heading.

| | |
|---|---|
| • SonarInterface.cc | • SonarInterface.h |
| • ArduinoInterface.cc | • ArduinoInterface.h |
| • DataPacket.cc | • DataPacket.hpp |
| • ArduinoCmds.hpp | • DataPacketStructs.hpp |

*Source List for Sonar Interface*

Source files for the RJSC01 system itself can also be found on the IGVC 2009 repository. Note that these files require the Arduino GUI environment to compile. More information can be found under the **RJSC01 Internal** section.

Below is an exhaustive function list for the RJSC01. All commands for the system are listed here.

| Name | Description |
|---|---|
| **Initialize System** | `void initSonDev(int num, int gain, int mrange, int freq);`<br><br>Initializes the RJSC01. This function takes in the number of SRFs in the array, the initial gain to assign to the SRFs, along with the initial maximum range value to assign all SRFs and the frequency of steps per second. This command should always be the first function called from this library. |
| **Ping Individual SRF** | `void pingIndiv(int id);`<br><br>Commands the specified SRF to ping. The ID value can be given as either the actual ID value (found on the back of each SRF, i.e. 0xF2) or indexed ID value (i.e. 0,1,2, etc). This command will allow the SRF to ping during it's step. If no other devices have been set to ping, it will do so immediately, otherwise it will wait until it's step number has been called. |
| **Read Individual SRF** | `int readIndiv(int id);`<br>`int readIndiv(int id, char type);`<br><br>Reads back the value of the specified SRF's most recent ping. It will return a -1 if the SRF has not finished pinging, otherwise it will return an integer value in millimeters. Optionally, the user can specify whether to grab the value from the RJSC01 itself or to return the last value that was recently grabbed from the system by specifying a 'n' or 'o' type value, respectively. This is useful in conjunction with the 'Update All' command. |
| **Ping All SRFs** | `void pingAll();`<br><br>Commands all active SRFs to ping in their specified step order. |
| **Update Range Data from All SRFs** | `int updateAll();`<br><br>Reads range data from all active SRFs. Note that this function does not return the data directly to the user. The range data can be fetched with the 'Read Individual SRF' command using the 'o' type. This function returns a -1 if one or more of the SRFs has not finished pinging, and returns a 0 if the command was completed successfully. |
| **Set Autopilot Mode On/Off** | `void setAutopilot(int on_off);`<br><br>Switches the RJSC01's autopilot mode to the on (1) or off (0) setting. Autopilot mirrors the 'Ping All SRFs' function, however every time the 'Update Range Data' function is called, the active SRF's are automatically set to re-ping. |
| **Set Gain for All SRFs** | `void setAllGain(int gainval);`<br><br>Assigns the specified gain value to all active SRF's in the array. Indoor applications typically require a lower gain, while outdoor applications call for larger values. The gain specified must take on an amount between 1 and 31. |
| **Set Gain for Individual SRF** | `void setIndivGain(int id, int gainval);`<br><br>Assigns the specified gain value to an SRF with a matching ID. Gain values must be between 1 and 31. |
| **Set Max Range Value for All SRFs** | `void setAllMRange(int mrangeval);`<br><br>Assigns the specified maxrange value to all active SRFs. This value dictates how long the SRF will spend listening for an echo. Shorter maxrange values yield a proportionally shorter ping time. The input takes on units of millimeters. |

| Name | Description |
|---|---|
| Set Max Range for Individual SRF | `void setIndivMRange(int id, int mrangeval);`<br><br>Assigns the specified maxrange value to an SRF with a matching ID. |
| Set Total Number of Steps | `void setStepTotal(int stepnum);`<br><br>Sets the total number of steps per loop. By default, this value is equal to the number of initialized SRFs. |
| Set Step Frequency | `void setFreq(int freqval);`<br><br>Sets the frequency value of steps per second. |
| Set Step ID for Individual SRF | `void setIndivStep(int id, int stepval);`<br><br>Sets the specified step ID to the desired SRF. By exploiting this function, different configurations of ping order can be achieved. It is legal to assign the same step ID to several SRFs. In this manner, all devices on this step will ping simultaneously. Note that step ID's can only take on values between 0 and the total step value less one. By default, each SRF is assigned an individual step ID. |
| Activate/Deactivate Individual SRF | `void setIndivActive(int id, int activeval);`<br><br>Activates or deactivates the specified SRF. Note that a deactivated SRF cannot ping or read back reliable distance values when deactivated. The activeval argument is set to 1 (on) or 0 (off). |
| Get Max Range Value from Individual SRF | `int getIndivMRange(int id);`<br><br>Returns the set maxrange value from the specified SRF. |
| Get Gain Value from Individual SRF | `int getIndivGain(int id);`<br><br>Returns the set gain value from the specified SRF. |
| Get Activation Status from Individual SRF | `int getIndivActive(int id);`<br><br>Returns a 1 if the specified SRF is active or a 0 otherwise. |
| Get Step ID from Individual SRF | `int getIndivStep(int id);`<br><br>Returns the step ID from the specified SRF. |
| Get Step Frequency | `int getFreq();`<br><br>Returns the step frequency value. |
| Get Total Step Number | `int getStepTotal();`<br><br>Returns the total number of steps per loop. |

**Code Samples**. Here is the standard setup for any RJSC01 interface application.

```
//==Standard Setup==//

#include “SonarInterface.h”
#include <stdlib.h>

//Create SonarInterface Object
SonarInterface sdev;

int main(void){
      //Call the Initializer:
      //Initialize 11 SRFs, with initial gain and maxrange of
      //12 and 2000 (resp), and a step frequency of 14.
      sdev.initSonDev(11,12,2000,14);

      //Place Your Code Here

}
```

The code sample above shows the skeleton of what a typical RJSC01 interface program should contain. In it, the SonarInterface library must be included. Then an object, arbitrarily named `sdev` in this block, of the SonarInterface type must be created, so that it can be manipulated and accessed throughout the main program. Inside main, a call to the SonDev initializer is called. From there, more code may be added to accommodate the user's specific application. Below is posted several samples of code for various other applications.

```
//Pinging an Individual SRF
#include "SonarInterface.h"
#include <stdlib.h>

SonarInterface sdev;

int main(void){

      int data;
      sdev.initSonDev(1,12,2000,14);

      //Ping the first SRF
      sdev.pingIndiv(0);

      //Get Range and Print it
      while((data = sdev.readIndiv(0)) == -1);
      printf("%d\n",data);

}
```

```
//Pinging an Array of SRFs
#include "SonarInterface.h"
#include <stdlib.h>

SonarInterface sdev;

int main(void){

      int data;
      sdev.initSonDev(16,12,2000,14);

      //Send Ping Command
      sdev.pingAll();

      //Request Range until it is Available
      while(sdev.updateAll() == -1);
      //Get Range and Print it
      for(int i=0;i<16;i++){
            //Print Value Grabbed from updateAll()
            printf("%d:%d\n",i,sdev.readIndiv(i,'o'));
      }

}
```

```
//Using Steps
#include "SonarInterface.h"
#include <stdlib.h>

SonarInterface sdev;

int main(void){

      int data;
      sdev.initSonDev(6,12,2000,14);

      //Set Up Steps:
      //3 Steps with a pair of SRFs pinging on each one
      sdev.setStepTotal(3);
      sdev.setIndivStep(0,0); sdev.setIndivStep(3,0);
      sdev.setIndivStep(1,1); sdev.setIndivStep(4,1);
      sdev.setIndivStep(2,2); sdev.setIndivStep(5,2);


      sdev.pingAll();

      while(sdev.updateAll() == -1);
      for(int i=0;i<6;i++){
            printf("%d:%d\n",i,sdev.readIndiv(i,'o'));
      }

}
```

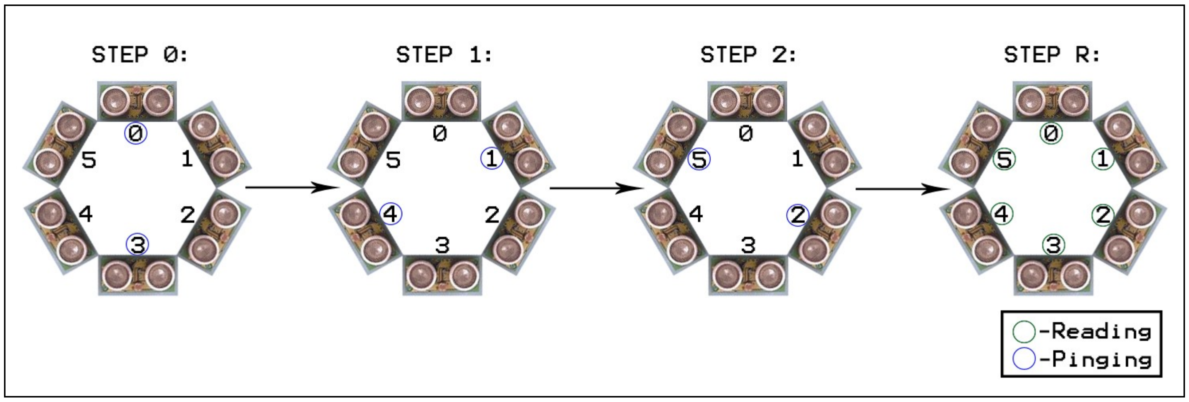


*Diagram of Step Sequencing*

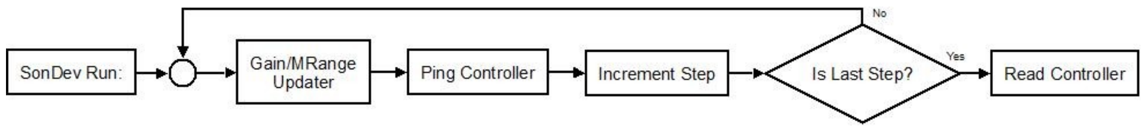
**RJSC01 Internal.** This system has been designed to control any moderately-sized array of SRF08 devices. Each SRF is assigned a SonDev structure, as detailed in the code block below. The address of a specific SRF is constant and is directly related to it's index number. By this convention, an SRF structure with an index value of 0 in the array will always communicate with a device with the address 0XE0, while an SRF structure of index 1 will always send commands to an addressed 0xE2 device. These addresses take on even values from 0xE0 (index 0) to 0xFE (index 15). The `addr` value in the SonDev structure takes on a value exactly *half* that of the device's actual address. This is for reasons concerning I2C communication protocol. Details can be found in the SRF08 datasheet or any I2C operation reference.

```
typedef struct _sondev{

	//addr: corresponds to the actual hex address used for direct
	//communication with the SRF over the I2C line. This is not
	//accessible to the user. Takes on values between 112 and 127. See
	//the SRF08 datasheet for details.
	byte addr;

	//gain: holds the current value of this SRF's gain.
	byte gain;

	//mrange: holds the current value of this SRF's max range.
	byte mrange;

	//seqid: holds the current value of this SRF's step identification
	//number.
	byte seqid;

	//rdata: holds the two-byte value of the most recently read range
	data.
	byte rdata[2];

	//flags: holds various flags for internal operation. These consist of
	//the following: FLAG_ACTIVE_SET(0x01), FLAG_CONT_SET(0x02),
	//FLAG_READ_SET(0x04), FLAG_PING_SET(0x08), FLAG_GEDIT_SET(0x10)
	//FLAG_MREDIT_SET(0x20)
	byte flags;

}SONDEV;
```

The `gain`, `mrange`, and `seqid` values in the SonDev structure are accessible to the user. The `rdata` variable is updated after all steps have been completed. Thus, if two SRFs are pinged, data will not be available for either until both have finished pinging. This `rdata` value is in units of centimeters (these are the default units of the SRF08) and is converted into millimeters within the SonarInterface's 'Read SRF' command.

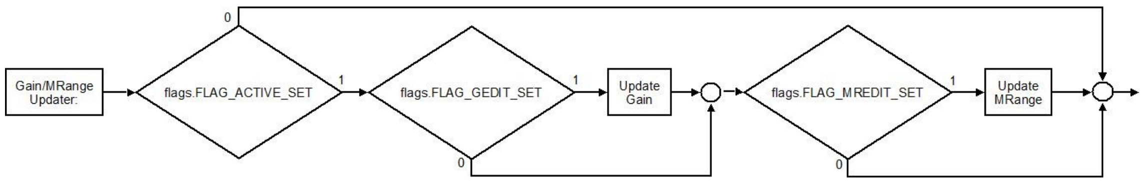
Various flags are used to control the flow of operation during the 'SonDev Run' command, internally specified by the RJSC01 software. These flags notify the system when an SRF is: active, in continuous mode, waiting to be read, waiting to ping, waiting to change its onboard `gain` value, and waiting to change its onboard `mrange` value.

This system runs an eternal loop that switches between checking for data on the USB port and running its control operations. USB communication consists of a header packet followed by a command message and its arguments, all handled by the SonarInterface library and the RJSC01 system itself. The control operations, also the 'SonDev Run' command, consist of the following segments: changing gain and/or max range values, pinging SRFs on the current step, incrementing the step, and (after all step sequences have finished) reading SRF range data. These processes are displayed in flowchart form, on the next page.

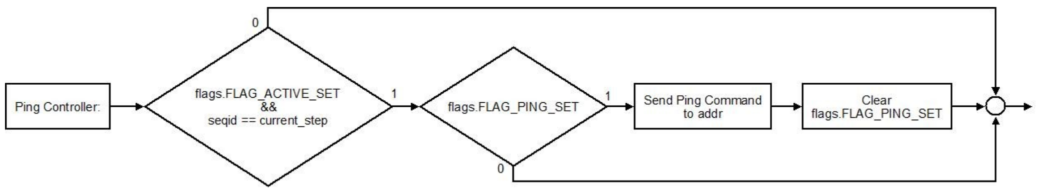


*SonDev Run: High Level Control Sequence*

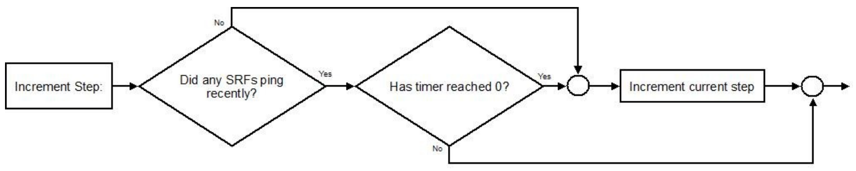
Note that the 'Gain/Mrange Updater', 'Ping Controller', and 'Read Controller' processes iterate for every SRF in the array (this number is set by the 'Initialize System' function) before moving on to the next process in line.



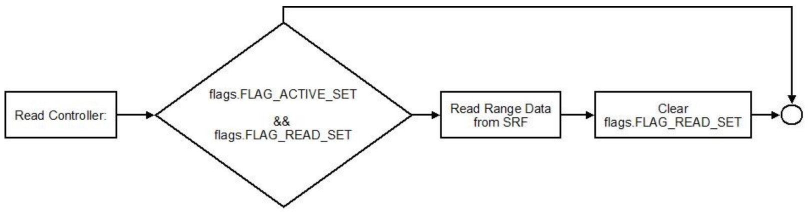
*Gain/MRange Updater: Control Sequence*


Ping Controller:
flags.FLAG_ACTIVE_SET && seqid == current_step
0
1
flags.FLAG_PING_SET
1
0
Send Ping Command to addr
Clear flags.FLAG_PING_SET

*Ping Controller: Control Sequence*



*Increment Step: Control Sequence*


Read Controller:
flags.FLAG_ACTIVE_SET && flags.FLAG_READ_SET
Read Range Data from SRF
Clear flags.FLAG_READ_SET

*Read Controller: Control Sequence*

Specifics on sending and receiving data over the I2C bus to an SRF can be found in the SRF08 datasheet, but is not detailed here. The 'SonDev Run' command utilizes a timer, as opposed to a delay, in order to provide a time gap between steps. This allows the user to continuously send and receive information from the RJSC01 without causing a program freeze during each step increment. Time gaps do not occur between steps if an SRF did not ping on that step. This eliminates time wasted on forcing a delay between each step. For example, if SRFs on step 0 and 3 are commanded to ping, the step sequences 1 and 2 will simply be skipped. In this example, only two time gaps occur: after step 0 and after step 3.

Unless malfunction of the Arduino Diecimila or it's Atmega168 controller occurs, use of the RJSC01's source files should not be needed. These files are, however, available on the Robojakcet's IGVC 2009 SVN. To compile them using the Arduino GUI environment, the user will need the following files in the project folder:

- RJSC01.pde
- sonar.h
- Arduino_SD.h
- twi.h
- sonar.c
- Arduino_SD.cpp
- twi.c

*Source List for RJSC01 board*

The RJSC01.pde file is used by the Arduino GUI as a main file. The sonar library consists of functions specific to SRF08 control and I2C communication. USB communication is handled by the Arduino_SD library. The twi source files deal with low level I2C functionalities.

**Changing SRF08's Physical Address.** In the event of an SRF malfunction, addition, or replacement, the address of the sensor will almost always need to be set to some desired value. By default, SRF08 modules are delivered by the manufacturer with an initial address of 0xE0. The following code (also available on the IGVC 2009 SVN) will perform the function of changing it's physical address. Do note that there should not be any other modules on the I2C line while this code is running. This file is to be compiled in the Arduino GUI environment.

```
//SonDev_AddressSet.pde
#include “Wire.h”
void Setup(){
      Wire.begin();
      Wire.beginTransmission(0x00);
      Wire.send(0x00);
      Wire.send(0xA0);
      Wire.endTransmission();
      Wire.begin();
      Wire.beginTransmission(0x00);
      Wire.send(0x00);
      Wire.send(0xAA);
      Wire.endTransmission();
      Wire.begin();
      Wire.beginTransmission(0x00);
      Wire.send(0x00);
      Wire.send(0xA5);
      Wire.endTransmission();
      Wire.begin();
      Wire.beginTransmission(0x00);
      Wire.send(0x00);
      //Change this line to the desired address(0xE0 thru 0xFE, even)
      Wire.send(0xF8);
      Wire.endTransmission();
}

void loop(){
}
```

After this code is loaded onto the Atmega168 controller and run, any devices on the I2C line will have their physical addresses set to 0xF8 (or whatever other value is chosen). Once the SRF's address is locked in, the red LED on the back of the device will switch on. At this time, the SRF is safe to remove from the I2C bus.