SRV-1-RCM User's Guide

SRV-1-RCM

Cautions

READ THE REQUIREMENTS SECTION BEFORE SUPPLYING VOLTAGE TO THE SRV-1-RCM.

Users of the Combo Matchport/Motor Driver board should note that although the RCM supplies operating voltage to the Matchport radio module and the Blackfin controller, it does not supply V+ or control signals to the H-bridge motor driver. Control can still be sent via the Blackfin but external V+ must be supplied to the Combo Matchport/Motor Driver board to power the motor drivers. BE VERY CAREFUL! If external power is supplied to the Combo Matchport/Motor Driver board the Matchport/Motor Driver board 3.3V regulator should be removed to prevent the external supply voltage from back-feeding into the RCM.

All the comm. ports run at 3.3v logic.

The analog and digital I/Os run a 5v supply through the center pin.

The motor ports can supply voltage internally (either 6v or 7v through the center pin depending on the J9 and J10 settings), or externally (check the setting of the P1 and P2 jumpers).

The RCM does not provide access to the following IOs of the Blackfin controller:

7/8/21/22 – (H-bridge controllers)
12 – (SPI_SEL3/SPI_Slave jumper)
17/23 – (GPIO-H0 Serial flow ctrl in/out - Matchport RTS0/CTSO (pins 7/11))
18 – (GPIO-H1 Ultrasonic module trigger (out))
19 – (GPIO-H2 Battery low-voltage detect (bat < 6V == hi))
24/26 – (GPIO-H7/H8 laser's)
27/28/29/30 – (GPIO-H10/H11/H12/H13 ultrasonic modules 1/2/3/4)
31/32 – (GPIO-H14/H15 (open))

Table of Contents

Cover Page
Cautions
SRV-1-RCM Introduction
RCM Feature Summary

Detailed Descriptions of Functions
- **Analog Inputs**
- **Digital GPIOs**
- **Motor/PPM Outputs**
- **Comm. Ports**
- **Micro SD Connector**

Requirements
SRV-1-RCM Diagram

SRV-1-RCM Introduction

The RCM is an expansion board for the Surveyor Corporation Blackfin Camera Module and is designed to bring extended robot control functions to the Blackfin Module. All the processing is handled by the Blackfin. The RCM is an interface that accepts sensor inputs, outputs PPM signals for motor control, and makes the communications options of the Blackfin easily accessible. The RCM also acts as the electrical supply interface for the Blackfin, the optional radio module available with the Blackfin, and all servo motors and sensors that connect to the RCM. The RCM also has onboard memory in the form of a micro SD slot.

RCM Feature Summary:

- 61 I/Os Total
- 16 Analog inputs (12 bit)
- 16 Digital GPIOs (8-bit) with integrated, programmable 100kΩ pull-up resistors
- 20 Servo/PPM outputs
- 2 SPI
- 2 UART
- 5 I2C
- Micro SD memory slot
- Built in power buses
- Small Size of 3.1”x 3.4”
- RoHS Compliant

Detailed Descriptions of Functions

Analog Inputs

The RCM employees two 8 port Analog Devices 7998 ADCs to allow access to sixteen analog sensors. The ADCs are I2C addressable:

Register 0x23 reads ports 11-18
Register 0x24 reads ports 21-28

However, with the use of the picoC programming language there are already pre-written functions to read the analog inputs; i.e. to read the sensor on port 14:

analog(14);

* Each read of an analog port is a single snapshot of what is on that port.

Connection to the analog ports is via a 3-pin connector. The pin pattern is: Ground/v+/Data. All analog devices are supplied with 5v through the center pin. If any connected analog device utilizes an external voltage source, the returned logic signal should still meet the 0-5v expectation of the AD7998. A common ground must also be accessed when external power is supplied to any analog device.

Digital GPIOs

The RCM employees a 16 port Microchip MCP23017 Digital GPIO with internal, programmable 100kΩ pull-up resistors. The MCP is also I2C addressable:

Register 0x27 is the MCP address
Register 0x00 sets Port A direction
Register 0x01 sets Port B direction
Register 0x12 reads or writes Port A bits
Register 0x13 reads or writes Port B bits

The following code sequence addresses the MCP:

writei2c(int, int, int);

The first parameter/argument takes the device I2C address

writei2c(0x27, int, int); talks to the MCP

The second parameter/argument sets which bank (A or B) is addressed

writei2c(0x27, 0x00, int); talks to Bank A
writei2c(0x27, 0x01, int); talks to Bank B

The third parameter/argument sets individual ports as input/output (1 for input, 0 for output).

Setting ports as input/output is done as a group with each port taking a binary place value. Since there are 8-ports per bank, each port takes on the identity of each bit in an 8-bit string. Port 1 equates to bit-1, Port 8 equates to bit-8.

Since binary is true/false (1/0), and input is set as true (1) and output is set as false (0); the value of the binary 8-bit position sets which port(s) is/are input or output.

So
writei2c(0x27, 0x00, 0x01); sets bit 1 (Port1) on Bank A to input and leaves bits 2-8 (Ports 2-8) as output.
writei2c(0x27, 0x00, 0x80); sets bit 8 on Bank A to input and leaves bits 1-7 as output.
writei2c(0x27, 0x00, 0x81); sets bits 1 and 8 on Bank A to input and leaves bits 2-7 as output.
writei2c(0x27, 0x00, 0xff); sets all bits on Bank A to input.
writei2c(0x27, 0x00, 0x8D); sets bits 1, 3, 4 and 8 on Bank A to input and leaves bits 2, 5, 6 and 7 as output.

* Replace the middle parameter/argument with 0x01 and Bank B is addressed.

Once the MCP is set for use, the following code will access the MCP to read/write to devices.

readi2c(0x27, 0x12); would read the Bank A ports
writei2c(0x27, 0x12, int); would write to the Bank A ports

readi2c(0x27, 0x13); would read the Bank B ports
writei2c(0x27, 0x13, int); would write to the Bank B ports

Read/Input

Physically the ports are numbered A1-8 and B1-8. When the read instruction is sent, all eight ports of the same bank are read together. The differentiation between the ports is the value they return.

Port 1 = 0/1
Port 2 = 0/2
Port 3 = 0/4
Port 4 = 0/8
Port 5 = 0/16
Port 6 = 0/32
Port 7 = 0/64
Port 8 = 0/128

If more than one port is positive/true, the result will be a sum total reading (i.e. 141 to represent ports 1, 3. 4 and 8 all being true).

* Note, the desired display form (binary, decimal, hexadecimal) can be specified in printf by use of the appropriate conversion character (%b, %d or %x).

Pull-Up Resistors

Sometimes a pull-up resistor is needed with digital inputs. The MCP has integrated 100kΩ pull-up resistors that can be programmed as on or off as needed.

To program the pull-up resistors:

Bank A
writei2c(0x27, 0x0c, int);

Bank B
writei2c(0x27, 0x0d, int);

The value placed in the int position will set the pull-ups on/off.

A (0) or false value sets the pull-up resistor off
A (1) or true value sets the pull-up resistor on

As explained in the Read/Input passage above, the actual value to be placed in int will be the bit value associated with the targeted port. For multiple ports the value placed in the int will be a sum total of the bits.

* Again, these pull-up resistors only apply to ports that are set as inputs.

Write/Output

All eight ports also write together. For output devices that need to be active a true (1) value needs to be sent. For output devices that need to be inactive a false (0) value needs to be sent.

Specifying the value by the binary position will set which ports are addressed. As above, with each port taking a binary place value, each port takes on the identity of its corresponding bit in an 8-bit string. Port 1 equates to bit-1, Port 8 equates to bit-8.

So
writei2c(0x27, 0x13, 0x01); writes a true (1) to bit 1 (Port 1) on Bank B and leaves bits 2-8 (Ports 2-8) as false (0).
writei2c(0x27, 0x13, 0x80); writes a true (1) to bit 8 on Bank B and leaves bits 1-7 as false (0).
writei2c(0x27, 0x13, 0x81); writes a true (1) to bits 1 and 8 on Bank B to input and leaves bits 2-7 as false (0).
writei2c(0x27, 0x13, 0xff); writes a true (1) to all bits on Bank B.
writei2c(0x27, 0x13, 0x8D); writes a true (1) to bitts 1, 3, 4 and 8 on Bank B and leaves bits 2, 5, 6 and 7 as false (0).

Connection to the digital ports is via a 3-pin connector. The pin pattern is: Ground/V+/Data. All digital devices are supplied with 5v through the center pin. If any connected digital device utilizes an external voltage source, the returned logic signal should still meet the 0-5v expectation of the MCP. A common ground must also be accessed when external power is supplied to any digital I/O device.

Motor/PPM Outputs

The RCM employees the 20 port Devantech SD20 servo driver chip. The SD20 is I2C addressable:

Register 0x61 is the SD20 address

writei2c( ) is a predefined function within picoC so controlling motors connected to the SD20 simply requires the function call with the appropriately set parameters. The following line of code will set the motor on port 14 of the SD20 to full speed CCW:

```
writei2c(0x61, 14, 0xff);
```

Notice how the address and setting are in hexadecimal, but the port number is the appropriate decimal associated with the port.

The SD20 outputs pulse widths from 1ms to 2ms in standard mode, but can output customized pulse widths in the expanded mode. The standard mode accommodates many motor controllers and servos across their entire range, but the expanded mode will accommodate many analog servos that only operate across a 90 degree sweep from 1ms to 2ms and need to range between 0.5ms and 2.5ms for 180 degrees of sweep.

* Again, to note; PPM signals are widely associated with positional servo motors, but many other types of motors can be driven using R/C PPM signals. Any motor controller that utilizes PPM wave patterns, whether they be for ant size motors or high voltage/high current motors can driven through the RCM. Effectively, the RCM with the SD20 can be used to control very small robots to multi-hundred pound goliath robots.

Motors connect to the SD20 ports via a 3-pin connector. The pin pattern is: Ground/V+/Data. Voltage to the ports can either be supplied internally or externally. The internal supply can be set to either 6v or 7v, and each bank of ten ports can supply up to 5A giving a total of 10A available for motor control. If this is not sufficient then external power can be supplied to the ports through a specified header.

Setting the SD20 (if needed)

Standard Mode

This is the power-up state with the servo range set in the standard 1mS to 2mS range. Writing any value greater than zero to Register 21 puts the SD20 into Expanded Mode. Writing a value of zero (0) to register 21 resets the SD20 to Standard Mode.

Expanded Mode

Expanded mode is used to manually set the PWM pattern of the SD20. PWM is pulse width modulation with the critical components being the width of each pulse, the range of pulse widths, and the center position, or offset of the pulse width range from zero.

* The duration between pulses is a standard for R/C type devices and permanently set in the SD20 so it does not need to be addressed in these settings.

Setting up expanded mode requires writing to three registers of the SD20 (21, 22 and 23). The following formulas and processes will explain how to determine the values to send to these registers.

* Be careful with this. Any PWM setting is used for all devices attached to the SD20, so if you introduce settings tuned toward the device with the widest range but have devices with a narrower range it is possible to cause the narrow range device to hit/stall at the stops and potentially burn out.

Register 21

Pulse Width Range (µs) = (255 * 256) / Reg21

Step one is to determine the range of possible pulse widths for the waveforms. This data is derived from the device(s) being used. Look at the operating range of each device. If the range is 1ms to 2ms you can operate in standard mode. If it is something like 0.70ms to 2.30ms then expanded mode is needed.

To determine the range of pulse widths needed simply subtract the narrowest pulse needed from the widest pulse needed. For our example 0.70ms to 2.30ms the difference (2.30-0.70) is 1.60ms.

Now, to derive the value to write to register 21, simply insert the value of the pulse width range determined above into the formula

Pulse Width Range (in µs) = (255 * 256) / Reg21

or

Reg21 = (255 * 256)/pulse width range (in µs)

* Note how this formula requires the pulse width range to be in microseconds. Most of the pulse width ranges for motors and motor controllers are listed in milliseconds. Be sure to convert milliseconds to microseconds (i.e. multiply by 1000).

  For a pulse width range of 1.60ms (1000)

Reg21 = (255 * 256)/1600 = **40.8**

Double checking, insert the new value of register 21 into the original formula.

* Since the register does not accept floating point numbers, the calculated value needs to be converted to an integer value. To keep from exceeding the allowable range of the servo it is best to round up.

Pulse Width Range (in µs) = (255 * 256) / 41 = 1592µs or 1.592ms

The value of 41 for register 21 is confirmed.

This value written to register 21 just sets the pulse width range. The range now needs to be shifted from starting at zero to actually starting at the specified low position (0.704 in our example). This requires an offset to be written to registers 22-23.

Registers 22-23
(Reg22:23 +20) (µs) = offset position

To calculate the offset simply set the equation equal to the offset position and solve.

* This formula also requires units in microseconds, so the offset position (the targeted low position) needs to be converted before applying to the equation (i.e. multiply ms by 1000)

For an offset position of 0.704ms (1000)

(Reg22:23 +20) (µs) = 704

(Reg22:23) = 704 - 20

(Reg22:23) = 684

All the registers of the SD20 are 8-bit. In an 8-bit system the highest number available is 255. The way to get a value like 684 takes two registers, thus 22:23. When splitting a number across two registers, one register becomes the high byte (Register 22 in this case) and one the low byte (Register 23 in this case). Also, it is best to use the 8-bit binary system to derive the full numerical value sought. So, with two registers, run two 8-bit strings together (like a 16-bit string) and derive the targeted value.

For 684

00000010 10101100

* If you are familiar with two-byte binary you may generate this type of answer on your own. If not, use a decimal to binary converter (which can be found with an internet search). Also, the binary values need to be converted to hexadecimal to write to the registers so use a binary to hexadecimal converter to get these values. When converting to hexadecimal convert each byte as its own single byte, 8-bit string.

00000010 is 2 in decimal and 0x02 in hexadecimal (this is the high byte/register 22)
10101100 is 172 in decimal and 0xAC in hexadecimal (this is the low byte/register 23)

So, to get the offset of 0.704ms:

Set Register 22 = 0x02
Set Register 23 = 0xAC

Writing the New Register Values
To write to the registers (21, 22 and 23) send the following code (based on the example above):

```
writei2c(0x61,21,0x41);
delay(100);
writei2c(0x61,22,0x02);
delay(100);
writei2c(0x61,23,0xAC);
delay(100);
```

Comm. Ports

The RCM interfaces directly to the comm. ports of the Blackfin giving access to two SPI comm.s, two UART comm.s, and five I2C access ports. All the comm. ports run at 3.3V logic; and addressing relevant to the Blackfin is exactly what is used to connect to the comm. ports of the RCM.

RCM	Blackfin
SPI 1 SEL	pin 13 (SPI_SEL2)
SPI 2 SEL	pin 20 (GPIO-H3)
Micro SD (SPI)	pin 25 (GPIO-H8)
UART 1*	pins 3/4
UART 2	pins 5/6
I2C (1-5)	pins 14/15

* The Matchport radio module connects to pins 3 and 4 of the Blackfin S32 header, so U1 is unavailable if the Matchport is used.

Micro SD Connector

There is currently a '$S' console command that can detect the presence and size of an SD card. Additional read/write commands will be added later.

Requirements

Input Voltage: 7.5V to 30V (A minimum of 6V can be used, but this will not be sufficient to power servo motors internally. However, if only motor controllers that require no voltage through the center pin are used then 6V is sufficient to drive the sensor ports, the SD20 control signal and the 3.3v logic supply to the Blackfin and optional radio module.)

DO NOT REVERSE THE CONNECTION TO THE INPUT VOLTAGE. BY DESIGN, TO ALLOW FOR LOW VOLTAGE SUPPLIES, NO REVERSE POWER SUPPLY PROTECTION WAS ADDED TO THE RCM.

The RCM must be connected to the Blackfin via S32 header (RCM supplies 3.3v logic to the Blackfin and optional radio module.)

Jumper blocks P1 and P2 must be in place for internal power to be supplied to the motor ports (P1 relates to motor ports 1-10, P2 relates to motor ports 11-20)

* If external power is required, remove P1 and/or P2 and feed external power at the P1/P2 header location (v+ to the center pin, ground to the outer pin)

DO NOT CONNECT EXTERNAL POWER TO THE INNER PIN (INTERNAL VOLTAGE SUPPLY PIN) ON P1/P2.

Jumper blocks J9 and J10 must be set to either 6v or 7v to regulate the power to the motor ports (J9 relates to motor ports 1-10, J10 relates to motor ports 11-20)

* Caution should be taken here to match the supply voltage to the motors/motor controller being used. Standard analog servos operate from 4.8v to 6v. The new digital robot servos operate from 6v to 7.2v. Running a digital robot servo at 6v will work fine, but higher speed and torque will be realized running at 7v. Running standard analog servos at 7v will also realize greater speed and torque out of the motors, but at 7v the motors are stressed and may burn out quickly.

NOTE
All processing is through the Blackfin Camera Module. Refer to the Blackfin user documents for instructions on Blackfin functions.
SRV-1 Blackfin Set-up (**http://www.surveyor.com/blackfin/SRV_setup_bf.html**)
SRV-1 Control Protocol (**http://www.surveyor.com/SRV_protocol.html**).

SRV-1-RCM Diagram