1.0 QS1R DDC Registers

All registers are 32 bits wide.

DDC_VERSION_REG (0x00) – read only																															
31	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**09**	**08**	**07**	**06**	**05**	**04**	**03**	**02**	**01**	**00**
0	0	0	0	0	1	1	1	0	0	0	1	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	1

Represents 0x07192011 – date of creation or version number.

DDC_CONTROL_REG_0 (0x01) – read/write																															
31	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**09**	**08**	**07**	**06**	**05**	**04**	**03**	**02**	**01**	**00**
MASTER RESET	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	**DAC CLK SELECT**	**WB BYPASS**	**DAC EXT MUTE EN**	**DAC BYPASS**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit 0	DAC Bypass	Set to 1 to bypass the DAC audio output.
Bit 1	DAC External Muting Enable	Set to 1 to enable external DAC muting input.
Bit 2	Wide Band Data Bypass	Set to 1 to bypass wideband data output on EP8.
Bit 3	DAC Clock Select	Set to 0 for 24 kSPS, set to 1 for 48 kSPS.
Bits 4 through 30	Not currently used.	
Bit 31	Master Reset	Set to 1 to hold DDC in reset.

Value on power up is 0x00000000.

DDC_CONTROL_REG_1 (0x02) – read/write																														
31	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**09**	**08**	**07**	**06**	**05**	**04**	**03**	**02**	**01**	**00**
NA	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	**DITHER**	**RANDOM**	**PGA**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit 0	ADC PGA Gain Select	Set to 0 for low gain, set to 1 for high gain.
Bit 1	ADC Randomizer Enable	Set to 1 to enable ADC randomizer function.
Bit 2	ADC Dither Enable	Set to 1 to enable ADC dithering function.
Bits 3 through 31	Not currently used.	

Value on power up is 0x00000000.

DDC_SAMPLE_RATE_REG – read/write			
Sample Rate		**Usable Bandwidth**	
25000	SPS	20	kHz
50000	SPS	40	kHz
125000	SPS	100	kHz
250000	SPS	200	kHz
625000	SPS	500	kHz
1250000	SPS	1.0	MHz
1562500	SPS	1.2	MHz
2500000	SPS	2.0	MHz

Writing other values than those above to the DDC sample rate register will cause a default to 50000 SPS.

Value on power up is 0x00000000.

GPIO_CONTROL_REG (0x03) – read/write																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
NA	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**GPIO14**	**GPIO13**	**GPIO12**	**GPIO11**	**GPIO10**	**GPIO9**	**GPIO8**	**GPIO7**	**GPIO6**	**GPIO5**	**GPIO4**	**GPIO3**	**GPIO2**	**GPIO1**	**GPIO0**
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Writing a 1 to the associated bit in this register causes the GPIO pin to become an input.
Writing a 0 to the associated bit in this register cause the GPIO pin to become an output.

Value on power up is 0xFFFFFFFF.

GPIO_IO_REG (0x04) – read/write																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
NA	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**GPIO14**	**GPIO13**	**GPIO12**	**GPIO11**	**GPIO10**	**GPIO9**	**GPIO8**	**GPIO7**	**GPIO6**	**GPIO5**	**GPIO4**	**GPIO3**	**GPIO2**	**GPIO1**	**GPIO0**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

On read, indicates whether the associated pin is low (0) or high (1).
On write, sets the associated pin low (0) or high (1).

Value on power up is 0x00000000.

RFB_CONTROL_REG (0x05) – read/write																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
NA	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**RFB11**	**RFB10**	**RFB9**	**RFB8**	**RFB7**	**RFB6**	**RFB5**	**RFB4**	**RFB3**	**RFB2**	**RFB1**	**RFB0**
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Writing a 1 to the associated bit in this register causes the RFB pin to become an input.
Writing a 0 to the associated bit in this register cause the RFB pin to become an output.

Value on power up is 0xFFFFFFFF.

RFB_IO_REG (0x06) – read/write																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
NA	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**RFB11**	**RFB10**	**RFB9**	**RFB8**	**RFB7**	**RFB6**	**RFB5**	**RFB4**	**RFB3**	**RFB2**	**RFB1**	**RFB0**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

On read, indicates whether the associated pin is low (0) or high (1).
On write, sets the associated pin low (0) or high (1).

Value on power up is 0x00000000.

EXT_CONTROL_REG (0x07) – read/write																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
NA	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**EXT19**	**EXT18**	**EXT17**	**EXT16**	**EXT15**	**EXT14**	**EXT13**	**EXT12**	**EXT11**	**EXT10**	**EXT9**	**EXT8**	**EXT7**	**EXT6**	**EXT5**	**EXT4**	**EXT3**	**EXT2**	**EXT1**	**EXT0**
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Writing a 1 to the associated bit in this register causes the EXT pin to become an input.
Writing a 0 to the associated bit in this register cause the EXT pin to become an output.

Value on power up is 0xFFFFFFFF.

EXT_IO_REG (0x08) – read/write																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
NA	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**NA**	**EXT19**	**EXT18**	**EXT17**	**EXT16**	**EXT15**	**EXT14**	**EXT13**	**EXT12**	**EXT11**	**EXT10**	**EXT9**	**EXT8**	**EXT7**	**EXT6**	**EXT5**	**EXT4**	**EXT3**	**EXT2**	**EXT1**	**EXT0**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

On read, indicates whether the associated pin is low (0) or high (1).
On write, sets the associated pin low (0) or high (1).

Value on power up is 0x00000000.

DDC_FREQRX0_REG (0x09) – read/write

Receiver 0 frequency control register.

register_value = desired_freq_hz / (encode_clock_freq_hz + correction_hz) * 4294967296

encode_clock_freq_hz = 125000000 (125 MHz) for all QS1R receivers.

Value on power up is 0x00000000.

DDC_FREQRX1_REG (0x0A) – read/write

Receiver 1 frequency control register. Not currently implemented. For future use.

DDC_FREQRX2_REG (0x0B) – read/write

Receiver 2 frequency control register. Not currently implemented. For future use.

DDC_FREQRX3_REG (0x0C) – read/write

Receiver 3 frequency control register. Not currently implemented. For future use.

2.0 Endpoint Descriptions and Data Formats

Endpoint 2 Data Format

Endpoint 2 is the audio DAC write channel. The audio DAC takes 16 bit L/R samples written to endpoint 2 and writes them to the DAC. The audio DAC can operate at either 24 kSPS or 48 kSPS by selecting the DAC clock in the **DDC_CONTROL_REG_0** (0x01), bit 3. The data to the DAC should be written in the following sequence: LRLRLR...

Endpoint 4 Data Format

Endpoint 4 is the FPGA configuration write channel. It is used by the QS1R firmware to transfer the FPGA configuration file in large blocks to the FPGA. This speeds up FPGA configuration. After the FPGA is configured, endpoint 4 is not used in normal operation.

Endpoint 6 Data Format

Endpoint 6 is the DDC data output read channel. The DDC decimates to the sample rate set in the DDC_SAMPLE_RATE_REG (0x03) and outputs 32 bit wide I and Q samples in the following sequence: QIQIQI...

Endpoint 8 Data Format

Endpoint 8 is the wide band data output read channel. The DDC buffers 16384, 16 bit data samples directly from the ADC and outputs them in a block on endpoint 8. This contains the 0 to 62.5 MHz spectrum.

3.0 Programming Examples

The following programming examples will use the Python qs_io.py module. You will need to have Python 2.7 installed. You will also need to install the pyusb module to use these examples.

Finding All QS1R Receivers on USB

```
import qs_io

qsio = qs_io.QsIO()
devs = qsio.find_devices()
if (len(devs) > 0 ):
        qsio.use_device(0)
```

The above example uses the first QS1R that is found on USB.

The QS1R Firmware

On power up, the QS1R does not contain any firmware. The firmware file and FPGA configuration file must be downloaded to the QS1R board. The Cypress Cy7C68013A microprocessor on the QS1R board contains a firmware loader function that must be used to download the QS1R firmware into the microprocessor's RAM.

```
success = qsio.load_firmware('qs1r_firmware_03032011.hex')
print success
```

If the firmware loads successfully, success will be True.

Once the firmware is loaded you can verify the firmware serial number with this function:

```
print qsio.read_fw_sn()
```


This should return '03032011' as the serial number.

The QS1R FPGA Configuration File

Once the firmware is loaded, you must next load the FPGA's configuration file.

```
success = qsio.load_fpga('QS1R_MASTER_3.rbf')
print success
```

If the FPGA file loaded successfully, success will be True.

You can read the FPGA configuration file's (DDC) version number with this function:

```
val = qsio.read_multibus_int(FPGARegisters.MB_DDC_VERSION)
print "FPGA Version Reg value: {0:08x}".format(val)
```

This should return '07192011' as the version number.

Reading and Writing the QS1R Configuration Registers

To read the DDC version number:

```
val = qsio.read_multibus_int(FPGARegisters.MB_DDC_VERSION)
print "FPGA Version Reg value: {0:08x}".format(val)
```

This should return '07192011' as the version number.

To read the **DDC_CONTROL_REG_0** register:

```
val = qsio.read_multibus_int(FPGARegisters.MB_CONTROL0)
print "Control Reg0 value: {0}".format(val)
```

To read the **DDC_CONTROL_REG_1** register:

```
val = qsio.read_multibus_int(FPGARegisters.MB_CONTROL1)
print "Control Reg1 value: {0}".format(val)
```

To read the **DDC_SAMPLE_RATE_REG** register:

```
val = qsio.read_multibus_int(FPGARegisters.MB_SAMPLERATE)
print "Sample Rate value: {0}".format(val)
```

To read the **DDC_FREQRX0_REG** register:

```
val = qsio.read_multibus_int(FPGARegisters.MB_FREQRX0)
print "Rx0 Freq Reg value: {0}".format(val)
```

To write the **DDC_SAMPLE_RATE_REG** register:

```
val = 125000
print qsio.write_multibus_int(FPGARegisters. MB_SAMPLERATE, val)
```

To read the **DDC_FREQRX0_REG** register:

```
val = 36627723
print qsio.write_multibus_int( MB_FREQRX0, val)
```

To set the **ADC PGA** to high:

```
val = qsio.read_multibus_int(MB_CONTROL1)
val |= MBControl1.PGA
print qsio.write_multibus_int(MB_CONTROL1, val)
```

To set the **ADC PGA** to low:

```
val = qsio.read_multibus_int(MB_CONTROL1)
val &= ~MBControl1.PGA
print qsio.write_multibus_int(MB_CONTROL1, val)
```

To set the **ADC Randomizer** on:

```
val = qsio.read_multibus_int(MB_CONTROL1)
val |= MBControl1.RAND
print qsio.write_multibus_int(MB_CONTROL1, val)
```

To set the **ADC Randomizer** off:

```
val = qsio.read_multibus_int(MB_CONTROL1)
val &= ~MBControl1.RAND
print qsio.write_multibus_int(MB_CONTROL1, val)
```

To read streaming IQ data from ep6:

```
length = 4096
buffer = qsio.read_ep6(length)
print buffer
```

To read wide band data from ep8:

```
length = 4096
buffer = qsio.read_ep8(length)
print buffer
```

Reading from endpoints:

```
buffer = qsio.read_ep6(length)

buffer = qsio.read_ep8(length)
```

Writing to endpoints:

```
print qsio.write_ep2(buffer)

print qsio.write_ep4(buffer)
```

There are functions in qs_io.py that read/write the I2C bus and the microprocessor EEPROM connected to the I2C bus:

```
result = qsio.read_i2c(address, length)
print qsio.write_i2c(address, values)

result = qsio.read_eeprom(address, offset, length)
print qsio.write_eeprom(address, offset, values)
```

These functions should be used with extreme care, especially the EEPROM write command. Under normal use, you should not have to write to the QS1R EEPROM. Doing so can void your warranty and cause the QS1R to become non responsive on the USB bus.

When writing to the I2C bus, avoid writing to the I2C address of the EEPROM which is 0x51.

For details of the GPIO, RFB, and EXT buses, see the QS1R Schematic.

For details of the ADC PGA, RAND, and DITHER functions see the LTC2208 Datasheet.