Celoxica

Platform Developer's Kit

RC200 Platform Support Library Reference Manual

For PDK v2.0

Authors: SB

Document number: RM-1050-1.1

Table of contents

Table of contents

Table of contents

Table of contents

Conventions

A number of conventions are used in this document. These conventions are detailed below.

Warning Message. These messages warn you that actions may damage your hardware.

Handy Note. These messages draw your attention to crucial pieces of information.

Hexadecimal numbers will appear throughout this document. The convention used is that of prefixing the number with '0x' in common with standard C syntax.

Sections of code or commands that you must type are given in typewriter font like this:

```
void main();
```

Information about a type of object you must specify is given in italics like this:

copy ***SourceFileName DestinationFileName***

Optional elements are enclosed in square brackets like this:

struct [type_Name]

Curly brackets around an element show that it is optional but it may be repeated any number of times.

string ::= "{***character***}"

Assumptions

This manual assumes that you:

- have used Handel-C or have the Handel-C Language Reference Manual
- are familiar with common programming terms (e.g. functions)
- are familiar with MS Windows

Omissions

This manual does not include description of RC200 hardware or installation. These are provided in the *RC200 Hardware and Installation Manual*. This is installed in ***`InstallDir`***`\PDK\Documentation\PSL\RC200\Manuals`.

1. Introduction

The RC200 Platform Support Library is provided as part of the Platform Developer's Kit. There are two slightly different versions:

- `rc200.hcl` targets the Standard and Professional versions of the RC200
- `rc200e.hcl` targets the Expert version of the RC200.

The library files are installed in ***InstallDir***`\PDK\Hardware\Lib\`, with the corresponding header files in ***InstallDir***`\PDK\Hardware\Include\`.

The RC200 Platform Support Library (PSL) simplifies the process of programming the FPGA to target the devices connected to it on the RC200 board. It also allows you to configure the FPGA from SmartMedia, and send data between the FPGA and host PC. The library conforms to the Platform Abstraction Layer (PAL) API, enabling you to write code that can easily be ported to other boards.

For information on the RC200 devices, refer to the RC200 Hardware and Installation Manual. This is installed in ***InstallDir***`\PDK\Documentation\PSL\RC200\Manuals`.

Using the library

Check that the DK library and include paths are set to ***InstallDir***`\PDK\Hardware\Lib` and ***InstallDir***`\PDK\Hardware\Include`. You can set these in the **Tools**>**Options**>**Directories** dialog in DK.

Before you include the library in your source code, you need to set the clock using one of the 4 preprocessor macros described in chapter 2: `RC200_CLOCK_USER`, `RC200_CLOCK_EXPCLK0`, `RC200_CLOCK_EXPCLK1` or `RC200_TARGET_CLOCK_RATE`.

After you have set the clock, include `rc200.hch` if you are targeting Standard or Professional boards, or `rc200e.hch` if you are targeting the Expert board.

For example, if you were targeting the Standard RC200 and wanted a clock rate of 50MHz:

```
# define RC200_TARGET_CLOCK_RATE = 50000000
#include "rc200.hch"
```

2. Clock definitions

To set the clock, you need to define one of the 4 preprocessor macros listed below, before including `rc200.hch` or `rc200e.hch` in your source code. If none of these are defined, no clock is set.

- `RC200_CLOCK_USER`
- `RC200_CLOCK_EXPCLK0`
- `RC200_CLOCK_EXPCLK1`
- `RC200_TARGET_CLOCK_RATE`

2.1 Specifying a clock source

```
# define RC200_CLOCK_USER

# define RC200_CLOCK_EXPCLK0

# define RC200_CLOCK_EXPCLK1
```

Description

To use CLKUSER (the FPGA clock) or one of the expansion header clocks, define one of the macros above before you include `rc200.hch` or `rc200e.hch` in your source code. The specified clock will be used by any subsequent `void main (void)` definition.

Defining `RC200_CLOCK_USER` will select the `CLKUSER` source from the clock generator. Defining `RC200_CLOCK_EXPCLK0` or `RC200_CLOCK_EXPCLK1` will select either `EXPCLK0` or `EXPCLK1` from the ATA expansion header.

2.2 Specifying a clock rate

```
# define RC200_TARGET_CLOCK_RATE
```

Description

To set a particular clock rate, use:

`# define RC200_TARGET_CLOCK_RATE =` ***`TargetRate`***

where ***`TargetRate`*** is the desired clock frequency in Hertz. A subsequent `void main (void)` definition will use a clock of approximately the desired frequency.

The actual frequency used will be returned in the macro `RC200_ACTUAL_CLOCK_RATE`. If `RC200_TARGET_CLOCK_RATE` is set to 24576000, 25175000, or 50000000 then the 24.576MHz, 25.175MHz or 50MHz on-board clocks will be used (respectively). Otherwise, a DCM will be used in frequency synthesis mode to generate the nearest approximation to the desired frequency (from a base of 50MHz). Note that the performance of generated clocks, in terms of parameters like jitter, may be worse than native clock frequencies. For more details about the DCM, consult the Xilinx Data Book.

Below 24MHz, Handel-C clock dividers will be used to divide the frequency down (since this is the lower bound of the DCM clock synthesis). This is handled transparently. The range of target frequencies is from 2MHz to 300MHz, but please note that the achievable frequency is design-dependent and will typically be much lower than 300MHz.

2.3 Checking the clock rate

`RC200_ACTUAL_CLOCK_RATE`

Description

You can define a target clock rate using the `RC200_TARGET_CLOCK_RATE()` macro. To determine the actual clock rate of your design, use the compile-time definition:

`RC200_ACTUAL_CLOCK_RATE`

3. Detecting the board type

```
extern macro expr RC200BoardIsExpert ();
```

Description

Returns a compile-time constant Boolean to indicate whether the board is an "Expert" model featuring expanded RAM, Bluetooth, LCD and touch screen.

You can use this to determine which board your code should be compiled for. For example, you could use an `if…select` statement to choose code specific to Expert boards.

4. LED macros

The LED macros target the blue LEDs on the RC200. The green LEDs on the RC200 are controlled by the CPLD and cannot be programmed.

To turn the blue LEDs on and off, you can either use `RC200LEDWrite()` and set ***Index*** to 0 to target LED0 or to 1 to target LED1, or you can use one of the `RC200LED*Write()` macros to target a specific LED. To control both LEDs at once, use `RC200LEDWriteMask`.

RC200LEDWrite()

`extern macro proc RC200LEDWrite (`***Index***`, `***Value***`);`

Parameters:	***Index***: LED index, of type unsigned 1.
	Value: Boolean control value, of type unsigned 1.
Timing:	1 clock cycle.
Description:	Turns the ***Index*** number LED either on or off. A ***Value*** of 1 means ON, and 0 means OFF.

RC200LED*Write() macros

`extern macro proc RC200LED0Write (`***Value***`);`

`extern macro proc RC200LED1Write (`***Value***`);`

Parameters:	***Value***: Boolean control value, of type unsigned 1
Timing:	1 clock cycle
Description:	Controls LED 0 or LED1. A ***Value*** of 1 means ON, and 0 means OFF.

LED write mask

`extern macro proc RC200LEDWriteMask (`***Value***`);`

Parameters:	***Value***: Bitmask control value, of type unsigned 2.
Timing:	1 clock cycle.
Description:	Controls both LEDs simultaneously. Bit 0 of ***Value*** controls LED 0, and bit 1 controls LED 1.

5. Push button macros

To test whether the buttons on or off, you can either use `RC200ButtonRead()` and set ***Index*** to 0 to test Button0 or to 1 to test Button1, or you can use one of the `RC200Button*Read()` macros to target a specific button. If you want to control both buttons at once, use `RC200ButtonReadMask()`.

RC200ButtonRead()

```
extern macro expr RC200ButtonRead (Index);
```

Parameters:	***Index***: Button index, of type `unsigned 1`.
Return value:	Boolean button state, of type `unsigned 1`.
Description:	Reads a value from either of the push buttons. A value of 1 means ON (or closed), a value of 0 means OFF (or open).

RC200Button*Read() macros

```
extern macro expr RC200Button0Read ();
```

```
extern macro expr RC200Button1Read ();
```

Parameters:	None.
Return value:	Boolean button state, of type `unsigned 1`.
Description:	Reads a value from push button 0 or 1.

RC200ButtonReadMask()

```
extern macro expr RC200ButtonReadMask ();
```

Parameters:	None.
Return value:	Bitmask of button state, of type `unsigned 2`.
Description:	Reads a value from both of the push buttons. The value at bit 0 is the state of button 0. The value at bit 1 is the state of button 1.

6. Seven-segment display macros

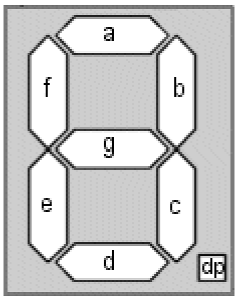
The seven-segment display macros allow you to write a specific hexadecimal digit to each display, or to specify which segments are lit up. SevenSeg0* macros target the left-hand display on the board and SevenSeg1* macros target the right-hand display.

6.1 Setting segments

```
extern macro proc RC200SevenSeg0WriteShape (Shape);

extern macro proc RC200SevenSeg1WriteShape (Shape);
```

Parameters:	***Shape***: Bitmask control value, of type unsigned 8.
Timing:	1 clock cycle.
Description:	Sets a particular shape in the seven-segment display. Shape is a binary mask where 1 means ON and 0 means OFF. Each of the eight bits corresponds to a segment of the display (7-segments for the digit and one for the decimal point). The segments are numbered as shown below. The right-most bit in ***Shape*** targets segment a, and the left-most bit targets the decimal point (dp).



Example

```
par
{
   RC200SevenSeg0WriteShape(11111100);
   RC200SevenSeg1WriteShape(01001111);
}
```

This would produce display "6.3" on the 7-segment display.

6. Seven-segment display macros

6.2 Writing digits

```
extern macro proc RC200SevenSeg0WriteDigit (Value, DecimalPoint);

extern macro proc RC200SevenSeg1WriteDigit (Value, DecimalPoint);
```

Parameters:	***Value***: Control value, of type `unsigned 4`.
	DecimalPoint: Control value, of type `unsigned 1`.
Timing:	1 clock cycle.
Description:	Sets a particular hex digit (0123456789abcdef) in the seven-segment display. ***Value*** is the hex value, and ***DecimalPoint*** specifies whether the decimal point should be turned on or off.

7. ZBT SRAM macros

If you want to read data from or write data to RAM you need to:

1. Call `RC200PL1RAM0Run()` or `RC200PL1RAM1Run()`, depending on which RAM bank you want to target. You need to call this in parallel with the rest of your RAM code.
2. Set the address for the read or write using one of the macros described in section 7.2.
3. Call one of the `RC200PL1RAM*Read()` or `RC200PL1RAM*Write()` macros.

If you only want to write part of a word of data, you can mask the address using one of the `RC200PL1RAM*SetWriteAddressMask ()` macros.

7.1 RAM management tasks

`extern macro proc RC200PL1RAM0Run (`***`ClockRate`***`);`

`extern macro proc RC200PL1RAM1Run (`***`ClockRate`***`);`

Parameters:	***ClockRate***: Clock rate of the clock domain of the call to this macro, in Hz.
Timing:	Does not terminate in normal use.
Description:	Runs the device management tasks for RAM. You must run this macro in parallel with accesses to the RAM banks.

7.2 Reading from and writing to RAM

7.2.1 Setting the RAM read and write addresses

`extern macro proc RC200PL1RAM0SetReadAddress (`***`Address`***`);`

`extern macro proc RC200PL1RAM1SetReadAddress (`***`Address`***`);`

`extern macro proc RC200PL1RAM0SetWriteAddress (`***`Address`***`);`

`extern macro proc RC200PL1RAM1SetWriteAddress (`***`Address`***`);`

Parameters:	***Address***: Address of data to read/write on the next clock cycle, of type `unsigned 19` on the Standard and Professional versions of the RC200, and `unsigned 20` on Expert boards.
Timing:	1 clock cycle.
Description:	Sets the address of data for the Read or Write which will occur on the next cycle.

Example

```
seq
{
   RC200PL1RAM0SetReadAddress (Addr);
   RC200PL1RAM0Read (&Data);
}
```

7.2.2 Write address mask

`extern macro proc RC200PL1RAM0SetWriteAddressMask (`***`Address`***`, `***`Mask`***`);`

`extern macro proc RC200PL1RAM1SetWriteAddressMask (`***`Address`***`, `***`Mask`***`);`

Parameters:	***Address***: Address of data to read/write on the next clock cycle, of type `unsigned 19` on the Standard and Professional RC200, and `unsigned 20` on Expert boards.
	Mask: data value of type `unsigned 4`.
Timing:	1 clock cycle.
Description:	Sets the address for the next write and masks the bytes that are set to 0 in ***Mask***. For example, if ***Mask*** was `0010`, only the second byte would be written to.

7.2.3 Reading from RAM

`extern macro proc RC200PL1RAM0Read (`***`DataPtr`***`);`

`extern macro proc RC200PL1RAM1Read (`***`DataPtr`***`);`

Parameters:	***DataPtr***: Pointer to an `lvalue` of type `unsigned 36`.
Timing:	1 clock cycle.
Description:	Reads a single item of data from the address specified by the call to the `RC200PL1RAM*SetReadAddress()` on the previous cycle.

7.2.4 Writing data to RAM

`extern macro proc RC200PL1RAM0Write (`***`Data`***`);`

`extern macro proc RC200PL1RAM1Write (`***`Data`***`);`

Parameters:	***Data***: Data value of type `unsigned 36`.
Timing:	1 clock cycle.
Description:	Writes a single item of data to the address specified by the call to `RC200PL1RAM*SetWriteAddress()` on the previous clock cycle.

8. PS/2 port macros

To write data to or read data from the mouse or keyboard, you need to:

1. Call `RC200PS2MouseRun()` or `RC200PS2KeyboardRun()`.
2. Call the appropriate read or write macro in parallel with this.

8.1 Mouse management tasks

```
extern macro proc RC200PS2MouseRun (ClockRate);
```

Parameters:	***ClockRate***: Clock rate of the clock domain of the call to this macro, in Hz.
Timing:	Does not terminate in normal use.
Description:	Runs the device management tasks for the mouse. You must run this macro in parallel with accesses to the device.

8.2 Reading data from the mouse

```
extern macro proc RC200PS2MouseRead (DataPtr);
```

Parameters:	***DataPtr***: Pointer to an `lvalue` of type `unsigned 8`.
Timing:	1 or more clock cycles (the read is blocked until data is ready).
Description:	Reads a single item of data from the mouse PS/2 port and stores it in the `lvalue` pointed at by ***DataPtr***. Note that these are raw bytes from the mouse. To do interpreted access (e.g. mouse positions) you should use the PAL PS/2 API. PAL documentation is in ***InstallDir***`\PDK\Documentation\PAL`.

8.3 Writing data to the mouse

```
extern macro proc RC200PS2MouseWrite (Data);
```

Parameters:	***Data***: Data value of type `unsigned 8`.
Timing:	1 or more clock cycles (until data is sent).
Description:	Writes a single item of data to the mouse PS/2 port from the expression ***Data***. Note that these are raw bytes to the mouse. To do interpreted access (e.g. mouse positions) you should use the PAL PS/2 API. PAL documentation is in ***InstallDir***`\PDK\Documentation\PAL`.

8. PS/2 port macros

8.4 Keyboard management tasks

`extern macro proc RC200PS2KeyboardRun (`***`ClockRate`***`);`

Parameters:	***`ClockRate`***: Clock rate of the clock domain of the call to this macro, in Hz.
Timing:	Does not terminate in normal use.
Description:	Runs the device management tasks for the keyboard. You must run this macro in parallel with accesses to the device.

8.5 Reading data from the keyboard

`extern macro proc RC200PS2KeyboardRead (`***`DataPtr`***`);`

Parameters:	***`DataPtr`***: Pointer to an `lvalue` of type `unsigned 8`.
Timing:	1 or more clock cycles (the read is blocked until data is ready).
Description:	Reads a single item of data from the keyboard PS/2 port and stores it in the `lvalue` pointed at by ***`DataPtr`***. Note that these are raw bytes from the keyboard. To do interpreted access (e.g. ASCII keyboard characters) you should use the PAL PS/2 API. PAL documentation is in ***`InstallDir`***`\PDK\Documentation\PAL`.

8.6 Writing data to the keyboard

`extern macro proc RC200PS2KeyboardWrite (`***`Data`***`);`

Parameters:	***`Data`***: data value of type `unsigned 8`.
Timing:	1 or more clock cycles (until data is sent).
Description:	Writes a single item of data to the keyboard PS/2 port from the expression ***`Data`***. Note that these are raw bytes from the keyboard. To do interpreted access (e.g. ASCII keyboard characters) you should use the PAL PS/2 API. PAL documentation is in ***`InstallDir`***`\PDK\Documentation\PAL`.

9. RS232 port macros

To read from or write to the RS232 port, you need to:

1. Call RC200RS232Run(). This sets the baud, parity, flow control and clock rate. Run this in parallel with the read or write macros.
2. Call RC200RS232Read() or RC200RS232Write().

9.1 RS232 management tasks

```
extern macro proc RC200RS232Run (BaudRate, Parity, FlowControl,
                                        ClockRate);
```

Parameters: ***BaudRate***: A code selecting the initial baud. Use the baud codes detailed section 9.1.1, RC200RS232SetBaudRate().

Parity: A code selecting the initial parity. Use the parity codes detailed in section 9.1.2, RC200RS232SetParity().

FlowControl: A code selecting the initial flow control. Use the flow codes detailed in section 9.1.3, RC200RS232SetFlowControl().

ClockRate: Clock rate of the clock domain of the call to this macro, in Hz.

Timing: Does not terminate in normal use.

Description: Runs the device management tasks for RS232 ports. Must always be run in parallel with accesses to the device.

9.1.1 Selecting the baud

```
extern macro proc RC200RS232SetBaudRate (BaudRate);
```

Parameters: ***BaudRate***: A code selecting the baud (see below).

Timing: 1 clock cycle.

Description: Changes the baud of the RS232 interface. ***BaudRate*** must be one of the codes listed below.

Baud code	Baud selected (number of transitions per second)
RC200RS232_75Baud	75
RC200RS232_110Baud	100
RC200RS232_300Baud	300
RC200RS232_1200Baud	1200

Baud code	**Baud selected (number of transitions per second)**
RC200RS232_2400Baud	2400
RC200RS232_9600Baud	9600
RC200RS232_19200Baud	19200
RC200RS232_38400Baud	38400
RC200RS232_57600Baud	57600
RC200RS232_115200Baud	115200
RC200RS232_230400Baud	230400
RC200RS232_460800Baud	460800
RC200RS232_921600Baud	921600

9.1.2 Selecting the parity

`extern macro proc RC200RS232SetParity (`***`Parity`***`);`

Parameters: ***`Parity`***: A code selecting the parity. Possible values:

`RC200RS232ParityNone`

`RC200RS232ParityEven`

`RC200RS232ParityOdd`

These correspond to the following settings: no parity bit; even parity bit; odd parity bit.

Timing: 1 clock cycle.

Description: Changes the parity setting of the RS232 interface.

9.1.3 Selecting the flow control

`extern macro proc RC200RS232SetFlowControl (`***`FlowControl`***`);`

Parameters:	***`FlowControl`***: A code selecting the flow control. Possible values:
	`RC200RS232FlowControlNone`
	`RC200RS232FlowControlSoft`
	`RC200RS232FlowControlHard`
	These correspond to the following settings: No flow control; Software flow control (XON/XOFF); Hardware flow (RTS/CTS)

Timing:	1 clock cycle.
Description:	Changes the flow control of the RS232 interface.

9.2 Reading from the RS232 port

`extern macro proc RC200RS232Read (`***`DataPtr`***`);`

Parameters:	***`DataPtr`***: Pointer to an `lvalue` of type `unsigned 8`.
Timing:	1 or more clock cycles (the read is blocked until data is ready).
Description:	Reads a single item of data from the RS232 port and stores it in the `lvalue` pointed at by ***`DataPtr`***.

9.3 Writing to the RS232 port

`extern macro proc RC200RS232Write (`***`Data`***`);`

Parameters:	***`Data`***: data value of type `unsigned 8`.
Timing:	1 or more clock cycles (until the data is sent).
Description:	Writes a single item of data to the RS232 port from the expression ***`Data`***.

10. Touch screen macros

You can use the touch screen macros to determine the position of the pointing device. `RC200TouchScreenReadRaw()` determines the position in raw coordinates. `RC200TouchScreenReadScaled()` determines the position scaled to 640 x 480 resolution.

10.1 Touch screen management tasks

`extern macro proc RC200TouchScreenRun (`***`ClockRate`***`);`

Parameters:	***ClockRate***: Clock rate of the clock domain of the call to this macro, in Hz.
Timing:	Does not terminate in normal use.
Description:	Runs the device management tasks for the touch screen. You must run this macro in parallel with accesses to the device.

10.2 Touch screen position (raw)

`extern macro proc RC200TouchScreenReadRaw (`***`XPtr`***, ***`YPtr`***, ***`TouchPtr`***`);`

Parameters:	***XPtr***: Pointer to an `lvalue` of type `unsigned 12`. ***YPtr***: Pointer to an `lvalue` of type `unsigned 12`. ***TouchPtr***: Pointer to an `lvalue` of type `unsigned 1`.
Timing:	1 clock cycle.
Description:	Returns the last sensed position of the pointing device on the touch screen, in raw coordinates. The coordinates range from 0 to 4095 and are independent of display resolution. The value returned in `*`***`TouchPtr`*** is the current state of the pointing device, where 1 means the pointer is touching the screen.

10.3 Touch screen position (scaled)

`extern macro proc RC200TouchScreenReadScaled (`***`XPtr`***`, `***`YPtr`***`, `***`TouchPtr`***`);`

Parameters:	***`XPtr`***: Pointer to an `lvalue` of type `unsigned 10`.
	`YPtr`: Pointer to an `lvalue` of type `unsigned 9`.
	`TouchPtr`: Pointer to an `lvalue` of type `unsigned 1`.
Timing:	1 clock cycle.
Description:	Returns the last sensed position of the pointing device on the touch screen, scaled to 640 x 480 resolution (the same as the LCD screen underneath). Note that the calibration of this scaling is only approximate; for precision use, each should be individually calibrated. The value returned in `*`***`TouchPtr`*** is the current state of the pointing device, where 1 means the pointer is touching the screen.

11. Video output macros

To use the video output macros, you need to:

1. Run `RC200VideoOutRun()` in parallel with the rest of your video output code.
2. Call `RC200VideoOutEnable()`.

11.1 Video output management tasks

`extern macro proc RC200VideoOutRun (`***`Mode`***`, `***`ClockRate`***`);`

Parameters:	***Mode***: Video mode expression, see below.
	ClockRate: Clock rate of the clock domain of the call to this macro, in Hz.
Timing:	Does not terminate in normal use.
Description:	Drives the video output in the selected mode. You must run this macro in parallel with accesses to the device. ***Mode*** must be one of the expressions listed below.
	The VGA modes drive the VGA connector with VESA GTF compatible timings. The horizontal resolution will adapt according to ***ClockRate***. To achieve standard resolutions, set the clock frequency to the appropriate value for the resolution, as shown in the table below.
	To use the LCD panel, the clock frequency must be exactly 25.175 MHz. When using the LCD the VGA connector is also driven, providing a dual display capability (although the image will be the same on both displays).

Mode expression	Video mode
`RC200VGAOutMode480at60`	480 lines at 60Hz refresh
`RC200VGAOutMode480at75`	480 lines at 75Hz refresh
`RC200VGAOutMode600at60`	600 lines at 60Hz refresh
`RC200VGAOutMode600at72`	600 lines at 72Hz refresh
`RC200VGAOutMode768at60`	768 lines at 60Hz refresh
`RC200VGAOutMode768at76`	768 lines at 76Hz refresh
`RC200VGAOutMode864at72`	864 lines at 72Hz refresh
`RC200VGAOutMode1024at75`	1024 lines at 75Hz refresh
`RC200LCDOutMode480at60`	480 lines at 60Hz refresh on LCD
`RC200TVOutModePAL`	PAL TV (625 lines interlaced @ 50Hz)
`RC200TVOutModeNTSC`	NTSC TV (525 lines interlaced @ 60Hz)

11. Video output macros

Resolution	Clock frequency
640 x 480 @ 60Hz	25.175000 MHz
640 x 480 @ 75Hz	31.500000 MHz
800 x 600 @ 60Hz	40.000000 MHz
800 x 600 @ 72Hz	50.000000 MHz
1024 x 768 @ 60Hz	65.000000 MHz
1024 x 768 @ 76Hz	85.000000 MHz
1152 x 864 @ 72Hz	100.000000 MHz
1280 x 1024 @ 75Hz	140.000000 MHz
720 x 576i @ 50Hz	13.846154 MHz
720 x 480i @ 60Hz	13.846154 MHz

11.2 Enabling video output

```
extern macro proc RC200VideoOutEnable ();
```

Parameters: None.

Timing: Typically 1 clock cycle.

Description: Enables the video output.

You need to call this macro before you call `RC200VideoOutWrite24()` or `RC200VideoOutWrite30()`.

11.3 Querying screen sizes

```
extern macro expr RC200VideoOutGetVisibleX (Mode, ClockRate);
extern macro expr RC200VideoOutGetVisibleY (Mode);
extern macro expr RC200VideoOutGetTotalX (Mode, ClockRate);
extern macro expr RC200VideoOutGetTotalY (Mode);
extern macro expr RC200VideoOutGetVisibleXCT (Mode, ClockRate);
extern macro expr RC200VideoOutGetVisibleYCT (Mode);
extern macro expr RC200VideoOutGetTotalXCT (Mode, ClockRate);
extern macro expr RC200VideoOutGetTotalYCT (Mode);
```

Y resolutions are independent of clock rate.

Parameters:	***Mode***: A video mode expression (see section 11.1).
	ClockRate: Clock rate of the clock domain of the call to `RC200VideoOutRun()` in Hz. Used to determine the horizontal screen resolution.
Description:	Macro expressions which return the dimensions of the visible screen (from 0 to `RC200VideoOutGetVisibleXY()-1`), and the total number of rows and columns scanned in including blanking.
	"CT" variants require a compile time constant mode, i.e. the ***Mode*** parameter must not be store in a variable or passed through a function parameter. As a result, the return value is also a compile time constant.

11.4 Disabling video output

```
extern macro proc RC200VideoOutDisable ();
```

Parameters:	None.
Timing:	Typically 1 clock cycle.
Description:	Disables the video output.

11.5 Writing a pixel

`extern macro proc RC200VideoOutWrite24 (`***`RGB24`***`);`

`extern macro proc RC200VideoOutWrite30 (`***`RGB30`***`);`

Parameters:	***RGB24***: Compound colour expression, of type `unsigned 24`.
	RGB30: Compound colour expression, of type `unsigned 30`.
Timing:	1 clock cycle.
Description:	Writes a single pixel to the display, at the current scan position.
	In both cases the video output expression is a concatenation of the red, green and blue components (i.e. R @ G @ B). In the case of 24-bit colour, these components are each 8 bits wide. In the case of 30-bit colour, these components are each 10 bits wide. In 24-bit mode, the lower DAC bits are suitably padded to use the entire output range.

11.6 Current scan position

```
extern macro expr RC200VideoOutGetX ();
```

```
extern macro expr RC200VideoOutGetY ();
```

Parameters:	None.
Description:	Macro expressions that return the current scan position of the screen output. A call to `RC200VideoOutWrite24()` or `RC200VideoOutWrite30()` will write a colour to the position on screen returned by these methods.

11.7 Blanking status of current scan position

```
extern macro expr RC200VideoOutGetHBlank ();
```

```
extern macro expr RC200VideoOutGetVBlank ();
```

Parameters:	None.
Description:	Macro expressions that return the horizontal or vertical blanking status of the current scan position, as type `unsigned 1`.

11.8 Horizontal and vertical sync status

```
extern macro expr RC200VideoOutGetHSync ();
```

```
extern macro expr RC200VideoOutGetVSync ();
```

Parameters:	None.
Description:	Macro expressions that return the horizontal or vertical sync status of the current scan position, as type `unsigned 1`.

12. Video input macros

There are 3 different macros for reading data:

- `RC200VideoInReadPixelPairYCrCb()` reads a pair of YCrCb pixels. YCrCb is the native output from the video decoder, and so this macro requires less hardware than the other two read macros.
- `RC200VideoInReadPixelPairRGB()` reads a pair of RGB pixels.
- `RC200VideoInReadPixelRGB()` reads a single RGB pixel.

Before you use one of these macros you need to:

1. Call `RC200VideoInRun()` in parallel with the rest of the video input code.
2. Select the type of video input using `RC200VideoInSetInput()`. If you do not set the input, Composite (CVBS) input will be used as a default
3. Select the colour–encoding standard using `RC200VideoInSetStandard()`. If you do not set the standard PAL/NTSC will be used by default.

12.1 Video input management tasks

`extern macro proc RC200VideoInRun (`***`ClockRate`***`);`

Parameters:	***`ClockRate`***: Clock rate of the clock domain of the call to this macro, in Hz.
Timing:	Does not terminate in normal use.
Description:	Runs the device management tasks for video input. You must run this macro in parallel with accesses to the video input device.

12.2 Selecting the video input

`extern macro proc RC200VideoInSetInput (`***`Input`***`);`

Parameters:	***`Input`***: A code selecting the video input. Possible values:
	`RC200VideoInInputComposite`
	`RC200VideoInInputCamera`
	`RC200VideoInInputSVideo`
	These codes correspond to the following inputs: Composite (CVBS) input; Camera input; S-Video input. The default value is `RC200VideoInInputComposite`.
Timing:	1 or more clock cycles.
Description:	Selects one of the three video inputs to sample.

12.3 Selecting the colour-encoding standard

`extern macro proc RC200VideoInSetStandard (`***`Standard`***`);`

Parameters: ***`Standard`***: A code selecting the TV colour-encoding standard. Possible values `RC200VideoInStandardPALNTSC` or `RC200VideoInStandardSECAM`

The first code selects PAL or NTSC and the second selects SECAM. The default value is `RC200VideoInStandardPALNTSC`.

Timing: 1 or more clock cycles.

Description: Selects which colour-encoding standard to expect at the selected input.

You need to call `RC200VideoInSetInput()` before using this macro. The video input is capable of decoding both PAL and NTSC from the same setting.

12.4 Reading a pair of YCrCb pixels

`extern macro proc RC200VideoInReadPixelPairYCrCb (`***`XPtr`***`, `***`YPtr`***`, `***`YCrCbPtr`***`);`

Parameters: ***`XPtr`***: Pointer to an `lvalue` of type `unsigned 9`.

`YPtr`: Pointer to an `lvalue` of type `unsigned 9`.

`YCrCbPtr`: Pointer to an `lvalue` of type `unsigned 32`.

Timing: 1 or more clock cycles (read blocks until data is ready).

Description: Reads a pair of YCrCb encoded pixels from the video input selected by `RC200VideoInSetInput()`. YCrCb is the native output from the video decoder and therefore requires the least hardware.

After the macro returns, (*`XPtr`, *`YPtr`) are the coordinates of the most recently sampled pixel, which has the colour value (*`YCrCbPtr`). Each pixel pair is presented at most once (pixels can be missed if they are not read quickly enough), at a rate of 6.75 MHz during the visible portion of the input video. The YCrCb data is formatted as follows:

(*YCrCbPtr)[31:24]	- Overall Cb (blue chrominance) value
(*YCrCbPtr)[23:16]	- Left-hand pixel Y (luminance) value
(*YCrCbPtr)[15: 8]	- Overall Cr (red chrominance) value
(*YCrCbPtr)[7: 0]	- Right-hand pixel Y (luminance) value

The chrominance and luminance values follow the CCIR601 standard ranges. The value in ***`XPtr`*** ranges from 0 to 718 (in 2-pixel increments) for an entire video line of 720 pixels. The value returned in ***`YPtr`*** varies from 0 to the number of visible lines – 1: 0-575 for PAL and 0-479 for NTSC.

12.5 Reading a pair of RGB pixels

```
extern macro proc RC200VideoInReadPixelPairRGB (XPtr, YPtr, LeftRGBPtr,
                                                RightRGBPtr);
```

Parameters: ***XPtr***: Pointer to an `lvalue` of type `unsigned 9`.

YPtr: Pointer to an `lvalue` of type `unsigned 9`.

LeftRGBPtr: Pointer to an `lvalue` of type `unsigned 24`.

RightRGBPtr: Pointer to an `lvalue` of type `unsigned 24`.

Timing: 1 or more clock cycles (read blocks until data is ready)

Description: Reads a pair of RGB encoded pixels from the video input selected by `RC200VideoInSetInput()`. This form of input requires a colour space converter which is built automatically. After the macro returns, (****XPtr***, ****YPtr***) are the coordinates of the most recently sampled pixel pair. The pair has the colour value (****LeftRGBPtr***, ****RightRGBPtr***). Each pixel pair is presented at most once, (pixels can be missed if they are not read quickly enough), at a rate of 6.75 MHz during the visible portion of the input video. The RGB data is formatted as follows:

(*LeftPtr or *RightPtr)[23:16] - Red value

(*LeftPtr or *RightPtr)[15: 8] - Green value

(*LeftPtr or *RightPtr)[7: 0] - Blue value

The chrominance and luminance values range from 0 to 255. The value in ***XPtr*** ranges from 0 to 718 (in 2-pixel increments) for an entire video line of 720 pixels. The value returned in ***YPtr*** varies from 0 to the number of visible lines - 1: 0-575 for PAL and 0-479 for NTSC.

12.6 Reading a single RGB pixel

`extern macro proc RC200VideoInReadPixelRGB (`***`XPtr`***`, `***`YPtr`***`, `***`RGBPtr`***`);`

Parameters: ***XPtr***: Pointer to an `lvalue` of type `unsigned 9`.

YPtr: Pointer to an `lvalue` of type `unsigned 9`.

RGBPtr: Pointer to an `lvalue` of type `unsigned 24`.

Timing: 1 or more clock cycles (read blocks until data is ready).

Description: Reads a single RGB encoded pixel from the video input selected by `RC200VideoInSetInput()`. This form of input requires a colour space converter which is built automatically. After the macro returns, (****XPtr***, ****YPtr***) are the coordinates of the most recently sampled pixel, which has the colour value (****RGBPtr***). Each pixel is presented at most once (pixels can be missed if they are not read quickly enough), at a rate of 13.5 MHz during the visible portion of the input video. The RGB data is formatted as follows:

(*RGBPtr)[23:16] - Red value

(*RGBPtr)[15: 8] - Green value

(*RGBPtr)[7: 0] - Blue value

The chrominance and luminance values range from 0 to 255. The value in ***XPtr*** ranges from 0 to 719 for an entire video line of 720 pixels. The value returned in ***YPtr*** varies from 0 to the number of visible lines - 1: 0-575 for PAL and 0-479 for NTSC.

13. Audio I/O macros

To use the audio macros you need to:

1. Call `RC200AudioRun()` in parallel with the rest of your audio code.
2. Set the audio input to the line in connector or the microphone using `RC200AudioInSetInput()`.

13.1 Audio codec management tasks

`extern macro proc RC200AudioRun (`***`ClockRate`***`);`

Parameters: ***ClockRate***: Clock rate of the clock domain of the call to this macro, in Hz.

Timing: Does not terminate in normal use.

Description: Runs the device management tasks for the Audio codec. You must run this macro in parallel with accesses to the device.

13.2 Setting the audio input

`extern macro proc RC200AudioInSetInput (`***`Input`***`);`

Parameters: ***Input***: Either `RC200AudioInLineIn` or `RC200AudioInMicrophone`.

Timing: 1 or more clock cycles.

Description: Sets the input of the audio ADC to be either the line in connector or the microphone.

13.3 Boosting the input amplifier

`RC200AudioInSetMicrophoneBoost (`***`Boost`***`);`

Parameters: ***Boost***: Data value of type `unsigned 2`.

Timing: 1 or more clock cycles.

Description: Sets the boost level of the microphone input amplifier, in +10dB steps, from 0 to +30dB.

13.4 Setting the gain level

```
extern macro proc RC200AudioInSetGain (Mute, LeftVol, RightVol);
```

Parameters: ***Mute***: Data value of type unsigned 1.

LeftVol: Data value of type unsigned 4.

RightVol: Data value of type unsigned 4.

Timing: 1 or more clock cycles.

Description: ***LeftVol*** and ***RightVol*** set the gain level (amount of increase) of the ADC input amplifiers, from 0dB to +22.5dB in 1.5dB steps. Mute is a Boolean where "1" = muted.

13.5 Setting the input sample rate

```
extern macro proc RC200AudioInSetSampleRate (SampleRateCode);
```

Parameters: ***SampleRateCode***: A code selecting the sampling rate. Possible values:

Sample rate code	Sample rate (Hz)
RC200AudioSampleRate8000	8000
RC200AudioSampleRate11025	11025
RC200AudioSampleRate16000	16000
RC200AudioSampleRate22050	22050
RC200AudioSampleRate32000	32000
RC200AudioSampleRate44100	44100
RC200AudioSampleRate48000	48000 (default)

Timing: 1 clock cycle.

Description: Changes the sample rate of the audio input.

13.6 Reading from the audio interface

`extern macro proc RC200AudioInRead (`***`LeftPtr`***`, `***`RightPtr`***`);`

Parameters:	***`LeftPtr`***: Pointer to an `lvalue` of type `signed 18`.
	`RightPtr`: Pointer to an `lvalue` of type `signed 18`.
Timing:	1 or more clock cycles (blocks until data is ready).
Description:	Reads a single stereo sample from the audio interface and stores it in the `lvalue` pointed at by ***`DataPtr`***. The macro blocks until a new sample can be read.

13.7 Setting the output volume

`extern macro proc RC200AudioOutSetVolume (`***`Mute`***`, `***`LeftVol`***`, `***`RightVol`***`);`

Parameters:	***`Mute`***: Data value of type `unsigned 1`.
	`LeftVol`: Data value of type `unsigned 5`.
	`RightVol`: Data value of type `unsigned 5`.
Timing:	1 or more clock cycles.
Description:	***`LeftVol`*** and ***`RightVol`*** set the gain level of the DAC output amplifiers, from 0dB to -46.5dB in -1.5dB steps. ***`Mute`*** is a boolean where "1" = muted.

13.8 Setting the output sample rate

`extern macro proc RC200AudioOutSetSampleRate (`***`SampleRateCode`***`);`

Parameters: ***`SampleRateCode`***: a code selecting the sampling rate. Possible values:

Sample rate code	**Sample rate (Hz)**
RC200AudioSampleRate8000	8000
RC200AudioSampleRate11025	11025
RC200AudioSampleRate16000	16000
RC200AudioSampleRate22050	22050
RC200AudioSampleRate32000	32000
RC200AudioSampleRate44100	44100
RC200AudioSampleRate48000	48000 (default)

Timing: 1 clock cycle.

Description: Changes the sample rate of the audio output.

13.9 Writing to the audio interface

```
extern macro proc RC200AudioOutWrite (Left, Right);
```

Parameters:	***Left***: Data value of type signed 20.
	Right: Data value of type signed 20.
Timing:	1 or more clock cycles (blocks until data is sent).
Description:	Writes a single stereo sample of data to the audio interface from the expressions ***Left*** and ***Right***. The macro blocks until a new sample can be written.

14. Bluetooth macros

To read from or write to the Bluetooth interface you need to:

1. Call `RC200BluetoothRun()`.
2. Call `RC200BluetoothRead ()` or `RC200BluetoothWrite()` in parallel with this.

You can reset the device using `RC200BluetoothReset()`.

14.1 Bluetooth management tasks

`extern macro proc RC200BluetoothRun (`***`ClockRate`***`);`

Parameters:	***`ClockRate`***: Clock rate of the clock domain of the call to this macro, in Hz.
Timing:	Does not terminate in normal use.
Description:	Runs the device management tasks for the Bluetooth interface. You must run this macro in parallel with accesses to the device.

14.2 Resetting the Bluetooth device

`extern macro proc RC200BluetoothReset ();`

Parameters:	None.
Timing:	1 or more clock cycles.
Description:	Resets the Bluetooth interface device.

14.3 Reading from the Bluetooth device

`extern macro proc RC200BluetoothRead (`***`DataPtr`***`);`

Parameters:	***`DataPtr`***: Pointer to an `lvalue` of type `unsigned 8`.
Timing:	1 or more clock cycles (read blocks until data is ready).
Description:	Reads a single item of data from the Bluetooth interface and stores it in the `lvalue` pointed at by ***`DataPtr`***. Note that these are raw bytes from the Bluetooth interface device. By default the Bluetooth interface device uses the BlueCore Serial Protocol (BCSP) from Cambridge Silicon Radio.

14.4 Writing to the Bluetooth device

```
extern macro proc RC200BluetoothWrite (Data);
```

Parameters:	***Data***: Data value of type unsigned 8.
Timing:	1 or more clock cycles (blocks until data is sent).
Description:	Writes a single item of data to the Bluetooth interface from the expression ***Data***. Note that these are raw bytes to the Bluetooth interface device. By default the Bluetooth interface device uses the BlueCore Serial Protocol (BCSP) from Cambridge Silicon Radio.

15. SmartMedia macros

The RC200 supports SmartMedia devices between 4 and 128 megabytes. For devices of 16 megabytes or more, you can use physical or logical addressing. You are recommended to use logical addressing, as this preserves the CIS block and misses out bad blocks. For devices of less than 16 megabytes, you can only use physical addressing.

Accessing the SmartMedia card

To use the RC200 PSL macros to access SmartMedia, you need to:

1. Call the `RC200SmartMediaRun()` macro in parallel with the other SmartMedia macros and in parallel to `RC200CPLDRun()`.
2. Enable the CPLD using `RC200CPLDEnable()`.
3. Call `RC200SmartMediaInit()` in parallel with `RC200SmartMediaRun()`, and before any of the other SmartMedia macros.

For example:

```
par
{
   RC200CPLDRun();
   RC200SmartMediaRun();
   seq
   {
      RC200CPLDEnable();
      RC200SmartMediaInit();
   }
}
```

Reading from or writing to SmartMedia

You are advised not to mix logical and physical addressing when accessing the SmartMedia card.

To perform a read or write using logical addressing you need to:

1. Call `RC200SmartMediaCheckLogicalFormat()`.
   If this macro returns 1 to indicate failure you need to perform a logical format on the card using the Celoxica FTU2 program (see section 15.7).
2. Set the address, using `RC200SmartMediaSetLogicalAddress()`.
3. Call `RC200SmartMediaRead()` or `RC200SmartMediaWrite()` for each byte of data. For the last byte of data, set the ***`LastData`*** compile-time constant to 1.
4. Call `RC200SmartMediaOperationEnd()` to complete the read or write process after all the data has been read or written.

15. SmartMedia macros

To perform a read or write using physical addressing you need to:

1. Set the address, using `RC200SmartMediaSetAddress()`.
2. Call `RC200SmartMediaRead()` or `RC200SmartMediaWrite()` for each byte of data. For the last byte of data, set the ***LastData*** compile-time constant to 1.
3. Call `RC200SmartMediaOperationEnd()` to complete the read or write process after all the data has been read or written.

Do not use the SmartMedia macros at the same time as any other accesses to the CPLD. If you have called `RC200SmartMediaSetLogicalAddress()`, `RC200SmartMediaSetAddress()`, `RC200SmartMediaRead()` or `RC200SmartMediaWrite()`, you need to get the return value from `RC200SmartMediaOperationEnd()` before accessing the CPLD again. If you have used any of the other SmartMedia macros, you can access the CPLD after they have completed.

15.1 SmartMedia management tasks

extern macro proc RC200SmartMediaRun (***ClockRate***);

Parameters:	***ClockRate***: Clock rate of the clock domain of the call to this macro, in Hz.
Timing:	Does not terminate in normal use.
Description:	Runs the SmartMedia physical layer driver. You must run this macro in parallel with the CPLD controller macro, `RC200CPLDRun()`. The SmartMedia can only function once you have enabled the CPLD (see section 18.2). You must ensure that communications with SmartMedia are not run in parallel with other CPLD control commands.

15.2 Initializing the SmartMedia device

extern macro proc RC200SmartMediaInit (***ResultPtr***);

Parameters:	***ResultPtr***: Pointer to register of type unsigned 1. Returns 0 for success, 1 for failure.
Timing:	240 clock cycles or more.
Description:	Initializes the SmartMedia device and controller. You must call this macro when the board is switched on, or when a SmartMedia card is inserted. It performs a reset of the device and reads the ID to identify the size of the card. The ID read returns a Maker and Device code which the controller stores internally.

15.3 SmartMedia manufacturer and device code

```
extern macro expr RC200SmartMediaGetMakerCode ();
```

```
extern macro expr RC200SmartMediaGetDeviceCode ();
```

The manufacturer and device code of a SmartMedia device can be determined after a successful call to `RC200SmartMediaInit()` by calling `RC200SmartMediaGetMakerCode()` and `RC200SmartMediaGetDeviceCode()`. Both return values of type `unsigned 8`. For example:

```
unsigned 8 Maker, Device;
Maker = RC200SmartMediaGetMakerCode ();
Device = RC200SmartMediaGetDeviceCode ();
```

15.4 Resetting SmartMedia

extern macro proc RC200SmartMediaReset (***ResultPtr***);

Parameters:	***ResultPtr***: Pointer to register of type `unsigned 1`. Returns 0 for success, 1 for failure.
Timing:	110 clock cycles or more.
Description:	Resets the SmartMedia device. You can reset the device at any time; the reset operation is the only one that can be run ignoring the busy status returned by the SmartMedia device.

15.5 Erasing SmartMedia memory

extern macro proc RC200SmartMediaErase (***Address***, ***ResultPtr***);

Parameters:	***Address***: Block address in bytes of type `unsigned 27`. ***ResultPtr***: Pointer to register of type `unsigned 1`. Returns 0 for success, 1 for failure.
Timing	250 clock cycles or more.
Description:	Performs an erase on the entire block set by ***Address***. Note that for 16 pages per block (4/8MB cards) the block address resides in the top 18 bits. For cards with 32 pages per block, the block address is in the top 17 bits. You can check how many pages there are in a block in your card using `RC200SmartMediaIsBlock32Pages()`.

`RC200SmartMediaErase()` performs an erase on the entire block, regardless of page or column number.

15.6 Finding the number of pages per block

```
extern macro expr RC200SmartMediaIsBlock32Pages ();
```

You can determine whether your SmartMedia device has 16 or 32 pages per block by calling `RC200SmartMediaIsBlock32Pages()`. This expression returns a true condition for cards that are 32 pages per block, and a false denotes 16 pages. The expression will only return valid results after a successful call to `RC200SmartMediaInit()`.

15.7 Logical and physical addressing

The RC200 PSL allows you to perform reads and writes using physical or logical addressing. The advantages of using logical addressing are:

- It preserves the CIS (Card Information Structure) and IDI (ID information) fields. If you overwrite these fields, the SmartMedia card may not work with other hardware.
- It skips bad blocks, avoiding the risk of reading or writing invalid data.

You can only use logical addressing on cards of 16 megabytes or more.

To use logical addressing, you need to format the card using the command-line version of the Celoxica FTU2 program. This is described in the FTU2 User Guide in ***InstallDir***`\PDK\Documentation\PSL\FTU2`.

The logical formatting operation creates a logical address map on the third valid block in the card. This is to allow for corrupt blocks near the start of the card; the CIS/IDI fields are on the first valid block. For instance, if physical blocks 0 and 3 were corrupt, the SmartMedia card would have the following structure:

Block 0:	Corrupt
Block 1:	CIS/IDI (1st valid block)
Block 2:	Valid block (blank)
Block 3:	Corrupt
Block 4:	Logical map (3rd valid block)
Block 5:	Logical address 0 (1st valid block after the logical map)

You can use the PSL macro `RC200SmartMediaCheckLogicalFormat()` to check whether a card has been formatted with a Celoxica logical address map. To set a logical address, use `RC200SmartMediaSetLogicalAddress()`.

You can format a card for physical addressing, using `RC200SmartMediaFormat()`. To set a physical address, use `RC200SmartMediaSetAddress()`.

15. SmartMedia macros

For information on how the physical layer control works, or to target SmartMedia without using the PSL, refer to the RC200 Hardware and Installation Guide, the documentation for your SmartMedia card, or `http://www.ssfdc.or.jp/english/`.

15.7.1 Checking for a logical address map

`extern macro proc RC200SmartMediaCheckLogicalFormat (`***`ResultPtr`***`);`

Parameters:	***ResultPtr***: Pointer to register of type unsigned 1. Returns 0 if the card is correctly formatted with the Celoxica logical address map, or 1 if it is not.
Timing:	350 clock cycles or more.
Description:	This macro checks to see if the SmartMedia card is formatted according to the Celoxica logical address map. If it is, it returns 0 and stores the number of the block where the logical map is stored. If you then set a logical address to block 0, using `RC200SmartMediaSetLogicalAddress()`, this will target the first valid block after the logical address map (refer to the RC200 Hardware and Installation Guide for more detail). You must call this macro before using `RC200SmartMediaSetLogicalAddress()`.

15.7.2 SmartMedia Physical Specification

`extern macro proc RC200SmartMediaFormat (`***`ResultPtr`***`);`

Parameters:	***ResultPtr***: Pointer to register of type unsigned 1. Returns 0 for success, 1 for failure.
Timing:	250 clock cycles or more.
Description:	This macro checks to see if the SmartMedia card is formatted according to the SmartMedia Physical Specification. If the card is unformatted, it formats it. ***ResultPtr*** indicates whether the card has been successfully formatted or not. ***ResultPtr*** also returns 0 if the card was already formatted.

15.8 Reading from and writing to SmartMedia

15.8.1 Setting a logical address

`extern macro proc RC200SmartMediaSetLogicalAddress (`***`WriteNotRead`***`, `***`Address`***`);`

Parameters:	***WriteNotRead***: Compile time constant. To select a write, use 1. To select a read, use 0. ***Address***: Address in bytes, of type unsigned 27.
Timing:	Does not terminate in normal use unless a read or write and an operation end (`RC200SmartMediaOperationEnd`) are performed.
Description:	Sets the address for a SmartMedia read or write operation, using logical addressing. The only valid commands to follow this macro are `RC200SmartMediaRead()` or `RC200SmartMediaWrite()`. Ensure that that no other CPLD actions are carried out once an address has been set. The macro adjusts automatically for whether the address is in the first half of the page (address < 256), or the second half of the page (255 < address < 512).

15.8.2 Setting a physical address

`extern macro proc RC200SmartMediaSetAddress (`***`WriteNotRead`***`, `***`Address`***`);`

Parameters:	***WriteNotRead***: Compile time constant. To select a write, use 1. To select a read, use 0. ***Address***: Address in bytes, of type unsigned 27.
Timing:	Does not terminate in normal use unless a read or write and an operation end (`RC200SmartMediaOperationEnd`) are performed.
Description:	Sets the address for a SmartMedia read or write operation, using physical addressing. The only valid commands to follow this macro are `RC200SmartMediaRead()` or `RC200SmartMediaWrite()`. Ensure that that no other CPLD actions are carried out once an address has been set. The macro adjusts automatically for whether the address is in the first half of the page (address < 256), or the second half of the page (255 < address < 512).

15. SmartMedia macros

15.8.3 Reading from SmartMedia

```
extern macro proc RC200SmartMediaRead (DataPtr, LastData);
```

Parameters:	***DataPtr***: Register to store the data to be read, of type unsigned 8.
	LastData: Compile time constant to indicate the end of the data. Set ***LastData*** to 1 to indicate that the last byte of data is being read.
Timing:	160 clock cycles or more (including setting the address).
Description:	Reads one byte of data from the SmartMedia device.
	You need to call `RC200SmartMediaSetAddress()` before you call this macro. The data returned is from the first valid (non-corrupt) block after the address set by `RC200SmartMediaSetAddress()`. The read spans across blocks and will wrap to the beginning of the card if it is not terminated before this.
	When the last byte of data is being read, you should set ***LastData*** to 1 and then call `RC200SmartMediaOperationEnd()`.
	To perform a real-time check for errors whilst the read is in progress, use `RC200SmartMediaGetError()`.

15.8.4 Writing to SmartMedia

```
extern macro proc RC200SmartMediaWrite (Data, LastData);
```

Parameters:	***Data***: Register/value to write, of type unsigned 8.
	LastData: Compile time constant to indicate the end of the data. Set ***LastData*** to 1 to indicate that the last byte of data is being written.
Timing:	390 clock cycles or more (including setting the address).
Description:	Writes a byte of data to the SmartMedia card.
	You need to call `RC200SmartMediaSetAddress()` before you call this macro. You can terminate the write by setting ***LastData*** to 1.
	You must call `RC200SmartMediaOperationEnd()` at the end of the write.
	The write spans across the end of the card if it is not terminated before this. Data will be padded up to a page with "FF"s.
	To perform a real-time check for errors whilst the write is in progress, use `RC200SmartMediaGetError()`.

A write process erases the entire contents of the block, even if you only write one byte.

15.8.5 Checking for errors during reads and writes

```
extern macro expr RC200SmartMediaGetError();
```

You can use this macro to perform a real-time check for errors whilst a read or write operation is in progress. You can use the macro at any time after the first call to `RC200SmartMediaRead()` or `RC200SmartMediaWrite()` and before you have called `RC200SmartMediaOperationEnd()`.

15.8.6 Completing a read or write operation

```
extern macro proc RC200SmartMediaOperationEnd (ResultPtr);
```

Parameters:	***ResultPtr***: Pointer to register of type unsigned 1. Returns 0 for success, 1 for failure.
Timing:	1 or more clock cycles. It will take more than one clock cycle if you call the macro directly after the last call to `RC200SmartMediaRead ()` or `RC200SmartMediaWrite()`.
Description:	You can only call this macro after a call to `RC200SmartMediaRead()` or `RC200SmartMediaWrite()`. After a read or write is terminated, this macro ensures that all internal SmartMedia driver operations are terminated before the next command can be called. Both of the possible return values from this macro indicate that the SmartMedia driver is in an idle state. You can then carry out other non-SmartMedia CPLD operations.

16. Ethernet macros

Timing of the Ethernet macros is unpredictable, since the chip has its own CPU, and because of the unpredictable nature of network communications (for example, a packet could be corrupt).

1. Call `RC200EthernetRun()` in parallel with the rest of the read/write code.
2. Call `RC200EthernetEnable()`.
3. Call `RC200EthernetReadBegin()` or `RC200EthernetWriteBegin()`.
4. Call `RC200EthernetRead()` or `RC200EthernetWrite()`. Data is read or written one byte at a time.
5. Once the whole packet has been read or written, call `RC200EthernetReadEnd()` or `RC200EthernetWriteEnd()`.

16.1 Ethernet management tasks

`extern macro proc RC200EthernetRun (`***`ClockRate`***`, `***`MACAddress`***`);`

Parameters:	***ClockRate***: Clock rate of the clock domain of the call to this macro, in Hz. ***MACAddress***: Ethernet MAC address to be used by the Ethernet chip, of type `unsigned 48`.
Timing:	Does not terminate in normal use.
Description:	Runs the device management tasks for the Ethernet interface. You must run this macro in parallel with accesses to the device.

16.2 Enabling the Ethernet device

`extern macro proc RC200EthernetEnable (`***`Mode`***`);`

Parameters:	***Mode***: Specifies initialization settings for Ethernet interface. This should be set using one of the pre-defined modes. The only mode currently available is `RC200EthernetModeDefault`: `RC200EthernetEnable (RC200EthernetModeDefault);`
Timing:	1400 clock cycles or more. Blocks if the Ethernet device is not ready.
Description:	Takes the Ethernet device out of isolation mode, and programs the transmit and receive parameters according to ***Mode***. You must call this macro after `RC200EthernetRun()` and before you issue any other commands to the Ethernet chip. It must also be run after a call to `RC200EthernetDisable()`, to enable access to the Ethernet chip again.

16.3 Setting the Ethernet mode

```
extern macro expr RC200EthernetModeDefault;
```

This macro is used to set the mode for `RC200EthernetEnable()`. It commands the Ethernet device to come out of isolation mode, to enable receive and transmit functions and to turn on auto-negotiation for 100Mbit full-duplex.

16.4 Disabling the Ethernet device

```
extern macro proc RC200EthernetDisable ();
```

Parameters:	None.
Timing:	1400 clock cycles or more.
Description:	Puts the Ethernet device in isolation mode, and clears the transmit and receive parameters.

16.5 Resetting the Ethernet device

```
extern macro proc RC200EthernetReset ();
```

Parameters:	None.
Timing:	Dependant on clock rate. Minimum: 4 clock cycles.
Description:	Sets the reset pin low for at least 100ns, forcing the Ethernet chip to reset. You need to call `RC200EthernetEnable()` after you reset the device.

16.6 Reading a packet

16.6.1 Starting the read process

```
extern macro proc RC200EthernetReadBegin (StatusPtr, DestinationPtr,
        SourcePtr, DataByteCountPtr, ResultPtr);
```

Parameters:	***StatusPtr***: Pointer to data of type unsigned 16. Returns the status data from the received packet.
	DestinationPtr: Pointer to data of type unsigned 48. Returns the destination MAC address from the received packet.
	SourcePtr: Pointer to data of type unsigned 48. Returns the source MAC address from the received packet.
	DataByteCountPtr: Pointer to data of type unsigned 11. Returns the number of data bytes in the received packet.
	ResultPtr: Pointer to data of type unsigned 1. Returns 1 if the macro has timed out while waiting for a packet (failure) or 0 (success).
Timing:	At least 70 clock cycles. There is a timeout of 0.5s if no packet is received.
Description:	Checks if a packet is waiting to be read, and if it is, starts the read process and returns destination, source, status and byte count from the packet header.
	If it times out while waiting for a packet, ***ResultPtr*** is returned as '1', otherwise it is returned as '0'.

16.6.2 Reading a byte of data from a packet

```
extern macro proc RC200EthernetRead (DataPtr,ResultPtr);
```

Parameters:	***DataPtr***: Pointer to data of type unsigned 8. Returns a byte of data from the received packet.
	ResultPtr: Pointer to data of type unsigned 1. Returns 1 (failure) or 0 (success).
Timing:	2 or 7 clock cycles alternately, and up to 12 clock cycles for the final read.
	The macro reads a byte at a time, but Ethernet accesses are 16-bit. When a byte of data is already buffered on the chip the read only takes 2 clock cycles.
	Timing may differ if other accesses to the chip precede a read operation.

Description:	Reads a single data byte from the packet currently being read. Returns ***ResultPtr*** = 1 to indicate an error if there is no data remaining in the packet or if there is no read in progress. Data is read a byte at a time, but communications with the Ethernet chip are 16-bit, so a byte is buffered in the Ethernet data structure, until there are 16 bits to read. You must call `RC200EthernetReadBegin()` before this macro.

16.6.3 Completing the read process

```
extern macro proc RC200EthernetReadEnd (ResultPtr);
```

Parameters:	***ResultPtr***: Pointer to data of type `unsigned 1`. Returns 1 (failure) or 0 (success).
Timing:	7 clock cycles to 5ms, depending on the speed of response of the Ethernet device.
Description:	Completes the process of reading a packet from the Ethernet device. You must call this macro after all the data has been read from a packet.

16.7 Writing a packet of data to the network

16.7.1 Starting the write process

```
extern macro proc RC200EthernetWriteBegin (Destination, DataByteCount,
                                           ResultPtr);
```

Parameters:	***Destination***: Data of type `unsigned 48`. Specifies the destination MAC address for the packet. ***DataByteCount***: Data of type `unsigned 11`. Specifies the number of data bytes to be sent. Possible values: 64-1518. ***ResultPtr***: Pointer to data of type `unsigned 1`. Returns 1 (failure) or 0 (success).
Timing:	At least 100 clock cycles. Timing depends on what the chip is doing when the macro is called.
Description:	Starts a Packet Write operation to send data to the Ethernet device, and from there onto the network. If the Ethernet buffer is full of received (but unread) packets, the oldest one is dropped to make space for the new write packet.

16.7.2 Writing a byte of data to a packet

`extern macro proc RC200EthernetWrite (`***`Data`***`, `***`ResultPtr`***`);`

Parameters:	***Data***: Data of type unsigned 8, containing a byte of data to write to the packet. ***ResultPtr***: Pointer to data of type unsigned 1. Returns 1 (failure) or 0 (success).
Timing:	1 or 6 clock cycles alternately. This is because the macro writes a byte at a time, but Ethernet accesses are 16-bit. When a byte of data is already buffered on the chip the write only takes 1 clock cycle. Timing may differ if other accesses to the chip precede a write operation.
Description:	Writes a single byte of data to a packet. Returns ***ResultPtr*** = 1 to indicate an error if the expected number of bytes have already been written to the packet, or if there is no write in progress. Data is written a byte at a time, but communications with the Ethernet chip are 16-bit, so a byte is buffered in the Ethernet data structure, until there are 16 bits to write. You must call `RC200EthernetWriteBegin()` before this macro.

16.7.3 Completing the write process

`extern macro proc RC200EthernetWriteEnd (`***`StatusPtr`***`, `***`ResultPtr`***`);`

Parameters:	***StatusPtr***: Pointer to data of type unsigned 16. Returns the status data from the transmitted packet. ***ResultPtr***: Pointer to data of type unsigned 1. Returns 1 (failure – packet has not been transmitted) or 0 (success).
Timing:	45 clock cycles to 5ms (timeout), depending on speed of response of Ethernet device.
Description:	Completes the process of writing a packet, by commanding the Ethernet device to send it onto the network and waiting for completion or timeout. You must call this macro after all the data has been written to a packet.

17. Reconfiguring the FPGA

```
extern macro proc RC200Reconfigure (ImageAddress);
```

Parameters: ***ImageAddress***: Data of type `unsigned 16`, specifying the block address to start accessing the SmartMedia card at for reconfiguration.

Timing: If reconfiguration is success, the macro does not return.

Description: This macro reconfigures the FPGA from the SmartMedia.

You must run it in parallel with `RC200CPLDRun()`, and after calling `RC200CPLDEnable()`. It checks if a SmartMedia card is present, and if it is, it writes the SmartMedia block address to two CPLD registers, and then reads from another CPLD register which causes the CPLD to reconfigure the FPGA from that address. The address is passed in as a logical address, which is the physical address on the SmartMedia + 1. This allows for the CIS (Card Information Structure) block. If no SmartMedia card is present, the macro returns, otherwise it enters a loop until the FPGA is reconfigured.

18. CPLD control

You need to run the CPLD and enable it if you want to use:

- the SmartMedia macros (see chapter 15)
- the reconfiguration macro (chapter 17)
- the Send Protocol macros (chapter 19)

`RC200CPLDRun()` needs to be called in parallel to these macros, and `RC200CPLDEnable()` needs to be called before you access them.

18.1 CPLD management tasks

`extern macro proc RC200CPLDRun (`***`ClockRate`***`);`

Parameters:	***`ClockRate`***: Clock rate of the clock domain of the call to this macro, in Hz.
Timing:	Does not terminate in normal use.
Description:	Runs the device management tasks for the CLPD interface. You must run this macro in parallel with accesses to the CPLD.

18.2 Enabling the CPLD

`extern macro proc RC200CPLDEnable();`

Parameters:	None.
Timing:	2 cycles if CPLD is ready for use, otherwise, undetermined.
Description:	You need to call this macro in parallel with `RC200CPLDRun()`, and before accesses to the CPLD. The macro waits until the CPLD is ready and then sets the CPLD internal mode to normal operation. Refer to the RC200 Hardware and Installation Manual for more details.

Example

```
par
{
   RC200CPLDRun();
   seq
   {
      RC200CPLDEnable();
      // code for CPLD accesses
      …
   }
}
```

18. CPLD control

19. Transferring data between the FPGA and parallel port

The Send Protocol allows you to send data between the FPGA and your PC via the parallel port.

To write data to or read data from the host PC:

6. Call `RC200CPLDRun()` and `RC200CPLDEnable()` (see chapter 18).
7. Call `RC200SendProtocolEnable()` to enable the Send Protocol driver.
8. Call `RC200SendProtocolWrite()` or `RC200SendProtocolRead()`.

Do not use the Send Protocol macros at the same time as the SmartMedia macros.

19.1 Enabling the Send Protocol driver

```
extern macro proc RC200SendProtocolEnable();
```

Parameters:	None.
Timing:	1 clock cycle.
Description:	Enables the Send Protocol driver. You cannot use this at the same time as the RC200 SmartMedia macros.
	You need to call `RC200CPLDRun()` and `RC200CPLDEnable()` before calling this macro.
	You must call this macro before any calls to `RC200SendProtocolWrite()` or `RC200SendProtocolRead()`.

19.2 Disabling the Send Protocol driver

```
extern macro proc RC200SendProtocolDisable();
```

Parameters:	None.
Timing:	1 clock cycle.
Description:	Disables the Send Protocol driver.

19. Transferring data between the FPGA and parallel port

19.3 Writing data to the host PC

`extern macro proc RC200SendProtocolWrite(`***`Data`***`);`

Parameters:	***`Data`***: Data of type `unsigned 8` to be sent to the host PC.
Timing:	Variable. Depends on whether the host PC has read the previous data item.
Description:	Sends one byte of data from the FPGA to the host PC. This macro will block if the previous data item written has not yet been read by the host. You must call `RC200SendProtocolEnable()` before using this macro.

19.4 Reading data from the host PC

`extern macro proc RC200SendProtocolRead(`***`DataPtr`***`);`

Parameters:	***`DataPtr`***: Pointer to data of type `unsigned 8`, to return data read from the host PC.
Timing:	Variable. Depends on whether host PC has sent data to read.
Description:	Reads one byte of data from the host PC and writes it to the FPGA. This macro will block if the host has not sent any data to be read. You must call `RC200SendProtocolEnable()` before using this macro.

20. Expansion port pins

```
extern macro expr RC200ExpansionPins;
```

Description

Pin list for the ATA-style expansion connector on the RC200. You can use this to create your own interface to this connector.

Index

Index

Index

Customer Support at http://www.celoxica.com/support/

Celoxica in Europe	**Celoxica in Japan**	**Celoxica in Asia Pacific**	**Celoxica in the Americas**
T: +44 (0) 1235 863 656	T: +81 (0) 45 331 0218	T: +65 6896 4838	T: +1 800 570 7004
E: sales.emea@celoxica.com	E: sales.japan@celoxica.com	E: sales.apac@celoxica.com	E: sales.america@celoxica.com

www.celoxica.com