Gisselquist
Technology, LLC

REAL-TIME CLOCK SPECIFICATION

Dan Gisselquist, Ph.D.
dgisselq (at) opencores.org

June 2, 2015

Revision History

Rev.	Date	Author	Description
0.1	5/25/2015	Gisselquist	First Draft

Contents

Tables

Preface

Every FPGA project needs to start with a very simple core. Then, working from simplicity, more and more complex cores can be built until an eventual application comes from all the tiny details.

This real time clock is one such simple core. All of the pieces to this clock are simple. Nothing is inherently complex. However, placing this clock into a larger FPGA structure requires a Wishbone bus, and being able to command and control an FPGA over a wishbone bus is an achievement in itself. Further, the clock produces seven segment display output values and LED output values. These are also simple outputs, but still take a lot of work to complete. Finally, this clock will strobe an interrupt line. Reading and processing that interrupt line requires a whole 'nuther bit of logic and the ability to capture, recognize, and respond to interrupts. Hence, once you get a simple clock working, you have a lot working.

Dan Gisselquist, Ph.D.

1.

Introduction

This Real–Time Clock implements a twenty four hour clock, count-down timer, stopwatch and alarm. It is designed to be configurable to adjust to whatever clock speed the underlying architecture is running on, so with only minor changes should run on any fundamental clock rate from about 66 kHz on up to 250 TeraHertz with varying levels of accuracy along the way.

This clock offers a fairly full feature set of capability: time, alarms, a countdown timer and a stopwatch, all features which are available from the wishbone bus.

Other interfaces exist as well.

Should you wish to investigate your clock's stability or try to guarantee it's fine precision accuracy, it is possible to provide a time hack pulse to the clock and subsequently read what all of the internal registers were set to at that time.

When either the count–down timer reaches zero or the clock reaches the alarm time (if set), the clock module will produce an impulse which can be used as an interrupt trigger.

This clock will also provide outputs sufficient to drive an external seven segment display driver and 16 LED's.

Future enhancements may allow for button control and fine precision clock adjustment.

The layout of this specification follows the format set by OpenCores. This introduction is the first chapter. Following this introduction is a short chapter describing how this clock is implemented, Chapt. 2. Following this description, the Chapt. 3 gives a brief overview of how to operate the clock. Most of the details, however, are in the registers and their definitions. These you can find in Chapt. 4. As for the wishbone, the wishbone spec requires a wishbone datasheet which you can find in Chapt. 5. That leaves the final pertinent information necessary for implementing this core in Chapt. 6, the definitions and meanings of the various I/O ports.

As always, write me if you have any questions or problems.

2.

Architecture

Central to this real time clock architecture is a 48 bit sub–second register. This register is incremented every clock by a user defined 32 bit value, `CKSPEED`. When the register turns over at the end of each second, a second has taken place and all of the various clock registers are adjusted.

Well, not quite but almost. The 48 bit register is actually split into a lower 40 bit register that is common to all clock components, as well as separate eight bit upper registers for the clock, timer, and stopwatch. In this fashion, these separate components can have different definitions for when seconds begin and end, and with sufficient precision to satisfy most applications.

The next thing to note about this architecture is the format of the various clock registers: Binary Coded Decimal, or BCD. Hence an `8'h59` refers to a value of 59, rather than 89. In this fashion, setting the time to `24'h231520` will set it to 23 hours, 15 minutes, and 20 seconds. The only exception to this BCD format are the subseconds fields found in the stopwatch and time hack registers. Seconds and above are all encoded as BCD.

3.

Operation

3.1 Time

To set the time, simply write to the clock register the current value of the time. If the seconds hand is written as zero, subsecond time will be cleared as well. The new clock value takes place one clock period after the value is written to the bus.

To set only some parts of the time and not others, such as the minutes but not seconds or hours, write all '1's to the seconds and hours. In this way, writing a `24'h3f17ff` will set the minutes to 17, but not affect the rest of the clock.

This is also the way to adjust the display without adjusting time. Suppose you wish to switch to display option '1', just write a `32'h013fffff` to the register and the display will switch without adjusting time.

3.2 Count-down Timer

To use the count down timer, set it to the amount of time you wish to count down for. When ready, or even in the same cycle, enable the count–down timer by setting the RUN bit high. At this point in time, the count–down timer is running. When it gets to zero, it will stop and trigger an interrupt. You can tell if the alarm has been triggered by the TRIGGER bit being set. Any write to the timer register will clear the alarm condition.

While the timer is running, writing a '0' to the timer register will stop it without clearing the time remaining. In this state, writing to the register the RUN bit by itself will restart the timer, while anything else will set the timer to a new value. Further, if the timer is stopped at zero, then writing zero to the timer will reset the timer to the last start time it had.

3.3 Stopwatch

The stop watch supports three operations: start, stop, and clear. Writing a '1' to the stop watch register will start the stopwatch, while writing a '0' will stop it. When it starts next, it will start where it left off unless the register is cleared. To clear the register and set it back to zero, write a '2' to the register. This will effectively stop the register and clear it in one step. If the register is already stopped, writing a '3' will clear and start it in one step. However, the register can only be cleared while stopped. If the register is running, writing a '3' will have no effect.

3.4 Alarm

To set the alarm, just write the alarm time to the alarm register together with alarm enable bit. As with the time register, setting any field, whether hours, minutes, or seconds, to `8'hff` has no effect on that field. Hence, the alarm may be activated by writing `25'h13fffff` to the register and deactivated by writing `25'h03fffff`.

Once the alarm is tripped, the RTC core will generate an interrupt. Further, the tripped bit in the alarm register will be set. To clear this bit and the alarm tripped condition, either disable the alarm or write a '1' to this bit.

3.5 Time Hacks

For finer precision timing, the RTC module allows for setting a time hack and reading the value from the device. On the clock following the time hack being high, the internal state, to include the time and the 48 bit counter, will be recorded and may then be read out. In this fashion, it is possible to capture, with as much precision as the device offers, the current time within the device.

It is the users responsibility to read the time hack registers before a subsequent time hack pulse sets them to new values.

4.

Registers

This RTC clock module supports eight registers, as listed in Tbl. 4.1. Of these eight, the first four have been so placed as to be the more routine or user used registers, while the latter four are more lower level. Each register will be discussed in detail in this chapter.

Name	Address	Width	Access	Description
CLOCK	0	32	R/W	Wall clock time register
TIMER	1	32	R/W	Count–down timer
STPWTCH	2	32	R/W	Stopwatch control and value
ALARM	3	32	R/W	Alarm time, and setting
CKSPEED	4	32	R/W	Clock speed control.
HACKTIME	5	32	R	Wall clock time at last hack.
HACKCNTHI	6	32	R	Wall clock time.
HACKCNTLO	7	32	R	Wall clock time.

Table 4.1: List of Registers

4.1 Clock Time Register

The various bit fields associated with the current time may be found in the `CLOCK` register, shown in Tbl. 4.2. This register contains six clock digits: two each for hours, minutes, and seconds. Each

Bit #	Access	Description
28–31	R	Always return zero.
24–27	R/W	Seven Segment Display Mode.
22–23	R	Always return zero.
16–21	R/W	Current time, BCD hours
8–15	R/W	Current time, BCD minutes
0–7	R/W	Current time, BCD seconds

Table 4.2: Clock Time Register Bit Definitions

of these digits is encoded in Binary Coded Decimal (BCD). Therefore, 23 hours would be encoded

as 6'h23 and not 6'h17. Writes to each of the various subcomponent registers will set that register, unless the write value is a 8'hff. The behaviour of the clock when non–decimal values are written, other than all F's, is undefined.

Separate from the time, however, is the seven segment display mode. Four values are currently supported: 4'h0 to display the hours and minutes, 4'h1 to display the timer in minutes and seconds, 4'h2 to display the stopwatch in lower order minutes, seconds, and sixteenths of a second, and 4'h3 to display the minutes and seconds of the current time. In all cases, the decimal point will appear to the right of the lowest order digit and will blink with the second hand. That is, the decimal will be high for the second half of any second, and low at the top of the second.

4.2 Countdown Timer Register

The countdown timer register, whose bit–wise values are shown in Tbl. 4.3, controls the operation

Bit #	Access	Description
26–31	R	Unused, always read as '0'.
25	R/W	Alarm condition. Write a '1' to clear.
24	R/W	Running, stopped on '0'
16–23	R/W	BCD Hours
8–15	R/W	BCD Minutes
0–7	R/W	BCD Seconds

Table 4.3: Count–down Timer register

of the count–down timer. To use this timer, write some amount of time to the register, then write zeros with bit 24 set. The register will then reach an alarm condition after counting down that amount of time. (Alternatively, you could set bit 24 while writing the register, to set and start it in one operation.) To stop the register while it is running, just write all zeros. To restart the register, provided more than a second remains, write a `26'h1000000` to set it running again. Once the timer alarms, the timer will stop and the alarm condition will be set. Any write to the timer register after the alarm condition has been set will clear the alarm condition.

4.3 Stopwatch Register

The various bits of the stopwatch register are shown in Tbl. 4.4. Of note is the bottom bit that, when set, means the stop watch is running. Set this bit to '1' to start the stopwatch, or to '0' to stop the stopwatch. Further, while the stopwatch is stopped, a '1' can be written to the clear bit. This will zero out the stopwatch and set it back to zero.

4.4 Alarm Register

The various bits of the alarm register are shown in Tbl. 4.5. Basically, the alarm register consists a time and two more bits. The extra two bits encode whether or not the alarm is enabled, and whether

Bit #	Access	Description
24–31	R	Hours
16–23	R	Minutes
8–15	R	Sub Seconds
1–7	R	Sub Seconds
1	W	Clear
0	R/W	Running

Table 4.4: Stopwatch Register

Bit #	Access	Description
26–31	R	Always reads zeros.
25	R/W	Alarm tripped. Write a '1' to this register to clear any alarm condition. (A tripped alarm will not trip again.)
24	R/W	Alarm enabled
16–23	R	Alarm time, BCD hours
8–15	R	Alarm time, BCD minutes
0–7	R/W	Alarm time, BCD Seconds

Table 4.5: Alarm Register

or not it has been tripped. The alarm will be *tripped* whenever it is enabled, and the time changes to equal the alarm time. Once tripped, the alarm will stay in the alarmed or tripped condition until either a '1' is written to the tripped bit, or the alarm is disabled.

As with the clock and timer registers, writing eight ones to any of the BCD fields when writing to this register will leave those fields untouched.

4.5 Clock Speed Register

The actual speed of the clock is controlled by the `CKSPEED` register, shown in Tbl. 4.6. This register

Bit #	Access	Description
0–31	R/W	48 bit counter time increment

Table 4.6: Clock Speed Register

contains a simple 32 bit unsigned value. To step the clock, this value is extended to 48 bits and added to the fractional seconds value.

This value should be set to 2^{48} divided by the clock frequency of the controlling clock. Hence, for a 100 MHz clock, this value would be set to `32'd2814750`. For clocks near 100 MHz, this allows adjusting speed within about 40 clocks per second. For clocks near 500 MHz, this allows time adjustment to an accuracy of about about 800 clocks per second. In both cases, this is good enough

to maintain a clock with less than a microsecond loss over the course of a year. Hence, this RTC module provides more logical stability than most hardware clocks on the market today.

4.6 Time–hack time

To support finer precision clock control, the time–hack capability exists. This capability consists of three registers, the time–hack time register shown in Tbl. 4.7, and two registers (Tbls. 4.8 and 4.9)

Bit #	Access	Description
24–31	R	BCD Hours.
16–23	R	BCD Minutes.
8–15	R	BCD seconds.
0–7	R	Subseconds, encoded in 256ths of a second

Table 4.7: Time Hack Time Register

Bit #	Access	Description
0–31	R	Upper 32 bits of the internal 40 bit counter.

Table 4.8: Time Hack Counter, High

capturing the contents of the 40 bit internal counter at the time of the hack.

Bit #	Access	Description
24–31	R	Bottom 8 bits of the internal 40 bit counter.
0–23	R	Always read as '0'.

Table 4.9: Time Hack Counter, Low

The time–hack time register is perhaps the simplest to understand. This captures the time of the time–hack in hours, minutes, seconds, and 8 fractional subsecond bits. The top 24 bits of this register will match the bottom 24 bits of the clock time register at the time of the time hack. The bottom eight bits are the top eight bits of the 48 bit subsecond time counter. The rest of those 48 bits may then be returned in the other two time hack counter registers.

At present, this functionality isn't yet truly fully featured. Once fully featured, there will (should) be a mechanism for adjusting this counter based upon information gleaned from the hack time. Implementation details have to date prevented this portion of the design from being implemented.

5.

Wishbone Datasheet

Tbl. 5.1 is required by the wishbone specification, and so it is included here. The big thing to notice

Description	Specification
Revision level of wishbone	WB B4 spec
Type of interface	Slave, Read/Write
Port size	32–bit
Port granularity	32–bit
Maximum Operand Size	32–bit
Data transfer ordering	(Irrelevant)
Clock constraints	Faster than 66 kHz
Signal Names	Signal Name / Wishbone Equivalent: i_clk — CLK_I i_wb_cyc — CYC_I i_wb_stb — STB_I i_wb_we — WE_I i_wb_addr — ADR_I i_wb_data — DAT_I o_wb_ack — ACK_O o_wb_stall — STALL_O o_wb_data — DAT_O

Table 5.1: Wishbone Datasheet

is that this real time clock acts as a wishbone slave, and that all accesses to the clock registers are 32–bit reads and writes. The address bus does not offer byte level, but rather 32–bit word level resolution. Select lines are not implemented. Bit ordering is the normal ordering where bit 31 is the most significant bit and so forth. Although the stall line is implemented, it is always zero. Access delays are a single clock, so the clock after a read or write is placed on the bus the `i_wb_ack` line will be high.

6.

I/O Ports

The I/O ports for this core are shown in Tbls. 6.1 and Tbl. 6.2. Tbl. 6.1 reiterates the wishbone

Port	Width	Direction	Description
i_wb_cyc	1	Input	Wishbone bus cycle wire.
i_wb_stb	1	Input	Wishbone strobe.
i_wb_we	1	Input	Wishbone write enable.
i_wb_addr	5	Input	Wishbone address.
i_wb_data	32	Input	Wishbone bus data register for use when writing (configuring) the core from the bus.
o_wb_ack	1	Output	Return value acknowledging a wishbone write, or signifying valid data in the case of a wishbone read request.
o_wb_stall	1	Output	Indicates the device is not yet ready for another wishbone access, effectively stalling the bus.
o_wb_data	32	Output	Wishbone data bus, returning data values read from the interface.

Table 6.1: Wishbone I/O Ports

I/O values just discussed in Chapt. 5, and so need no further discussion here.

This clock is designed for command and control via the wishbone. No other registers, beyond the wishbone bus, are required. However, several other may be valuable. These other registers are listed in Tbl. 6.2. We'll discuss each of these in turn.

First of the other I/O registers is the `o_sseg` register. This register encodes which outputs of a seven segment display need to be turned on to represent the value of the clock requested. This register consists of four eight bit bytes, with the highest order byte referencing the highest order display segment value. In each byte, the low order bit references a decimal point. The other bits are ordered around the zero, with the top bit being the top bar of a '0', the next highest order bit and so on following the zero clockwise. The final bit of each byte, the bit in the two's place, encodes whether or not the middle line is to be displayed. When either timer or alarm is triggered, this display will blink until the triggering conditions are cleared.

This output is expected to be the input to a seven segment display driver, rather than being the output to the display itself.

The next output lines are the 16 lines of the `o_led` bus. When connected with 16 LED's, these lines will create a counting display that will count up to each minute, synchronized to the minute.

Port	Width	Direction	Description
o_sseg	32	Output	Lines to control a seven segment display, to be sent to that display's driver. Each eight bit byte controls one digit in the display, with the bottom bit in the byte controlling the decimal point.
o_led	16	Output	Output LED's, consisting of a 16–bit counter counting from zero to all ones each minute, and synchronized with each minute so as to create an indicator of when the next minute will take place when only the hours and minutes can be displayed.
o_interrupt	1	Output	A pulsed/strobed interrupt line. When the clock needs to generate an interrupt, it will set this line high for one clock cycle.
i_hack	1	Input	When this line is raised, copies are made of the internal state registers on the next clock. These registers can then be used for an accurate time hack regarding the state of the clock at the time this line was strobed.

Table 6.2: Other I/O Ports

When either timer or alarm has triggered, all of the LED's will flash together until the triggered condition is reset.

The third other line is the `o_interrupt` line. This line will be strobed by the RTC module any time the alarm is triggered or the timer runs out. The line will not remain high, but neither will it trigger a second time until the underlying interrupt is cleared. That is, the timer will only trigger once until cleared as will the alarm, but the alarm may trigger after the timer has triggered and before the timer clears.

The final other I/O line is a simple input line. This line is expected to be strobed for one clock cycle any time a time hack is required. For example, should you wish to read and synchronize to a GPS PPS signal, strobe the device with the PPS (after dealing with any metastability issues), and read the time hacks that are produced.