CHARA Technical Report

No. 74 4 June 1998

Local Clocks for Device Controllers

T.A. ten Brummelaar (CHARA)

1. INTRODUCTION AND GENERAL INFORMATION

The CHARA Array will employ five 1-m size, Alt/Az style telescopes at a site on Mount Wilson in southern California. The telescopes will be housed separately and operated remotely from a central laboratory. Light from each telescope will be directed by subsequent flat mirrors through vacuum pipes to additional optics and instrumentation at the central laboratory. More information about the CHARA Array can be found at our WWW home page listed below.

As set out in Technical Report TR68, it is critical for the multiple control computers in the array control system to function synchronously. This technical report describes the hardware and software interface required in each control computer to enable such synchronous operation.

2. THE MASTER CLOCK

The master clock will provide three different timing signals:

1. A 16 MHz pulse
2. A 1 mS pulse
3. A 1 S pulse

each of standard TTL voltages and assumed to go from a low state to a high state at the critical time. Line drivers will probably be required to distribute these signals around the OPLE/BCL building but will not be considered further in this report.

The 16MHz signal is provided only for the OPLE cart control system, a 'black-box' being supplied under contract by JPL. The OPLE electronics also require the 1-second tick in order to properly synchronize to the absolute time of the Array master clock. The OPLE system will not be discussed any further. All other controllers will use the 1-millisecond and 1-second ticks.

[1]Center for High Angular Resolution Astronomy, Georgia State University, Atlanta GA 30303-3083
TEL: (404) 651-2932, FAX: (404) 651-1389, FTP: ftp.chara.gsu.edu, WWW: http://www.chara.gsu.edu

As shown in TR68, the 16MHz will be used as a time standard and must be accurate to within one part in 10^9.

3. LOCAL CLOCKS

The local clocks are really just a means of getting the 1-millisecond and 1-second ticks into a PC running RT-Linux. The 1-millisecond ticks will cause an interrupt while the PC must poll for the 1-second tick.

For the sake of simplicity, and cheapness, the local clocks will use the standard PC parallel port. This port provides interrupt access (IRQ 7), and has several TTL inputs and outputs apart from the 8-bit port used for sending data. Thus, one TTL input will be used to fire interrupts and will connect to the 1-millisecond clock, a second TTL input will be used for the 1-second ticks, and the remaining three TTL inputs will be available for use by developers of individual control packages.

This is enough functionality for running the clock; however, since there are several input and output pins on the interface they may as well be made available to developers of control systems. Thus the 8-bit data port and two of the available TTL output pins will be used to implement two 8-bit digit-analog-converters (DACs). These DACs can be used by developers for almost anything they desire, but the intent is for aiding in the debugging of control software. When working on a digital servo it can be an invaluable aid to be able to see in real time what is going on. A variable in the software can be mapped to one of the DACs and it's behavior displayed on an oscilloscope. The two remaining TTL output pins on the port will also be available for use by developers.

The electronics will reside in a small metal box designed to fit inside a standard drive port in a PC. Furthermore, the electronics have been designed to use the same power supply as a standard disk drive so no separate power supply is required. The unit will have the same standard power supply socket as used in disk drives.

The connections to the outside world will be a D25 socket for connecting to the parallel port and BNC connectors for the rest of the signal inputs and outputs. All of these plugs will be mounted in bus slot covers so that they will be accessible from the rear of the computer.

The input and output parts of the local clocks will now be discussed separately.

3.1. Input Electronics and Connections

A schematic diagram of the input electronics is shown in Figure 1 and the mapping of input signals to the standard parallel port signal names is given in Table 1.

In order to protect the computer electronics, buffers are used in all signal lines. Thus if a dangerous voltage is sent into the local clock, the clock electronics will be damaged and not the computer mother-boards. The same will be true for the output electronics discussed in the next section.

Apart from the buffers, the input signals all connect directly onto the 25-D socket, except for the 1-second clock signal which is routed through a bipolar one-shot chip (74LS122). This chip creates a TTL pulse 0.5 milliseconds long when it sees it's input go from a low to a high state. Thus the 1-second clock tick will be visible by the control software for this period after the interrupt caused by the 1-millisecond tick. This is necessary because only one interrupt is available in the port and so the software will need to be able to poll for the

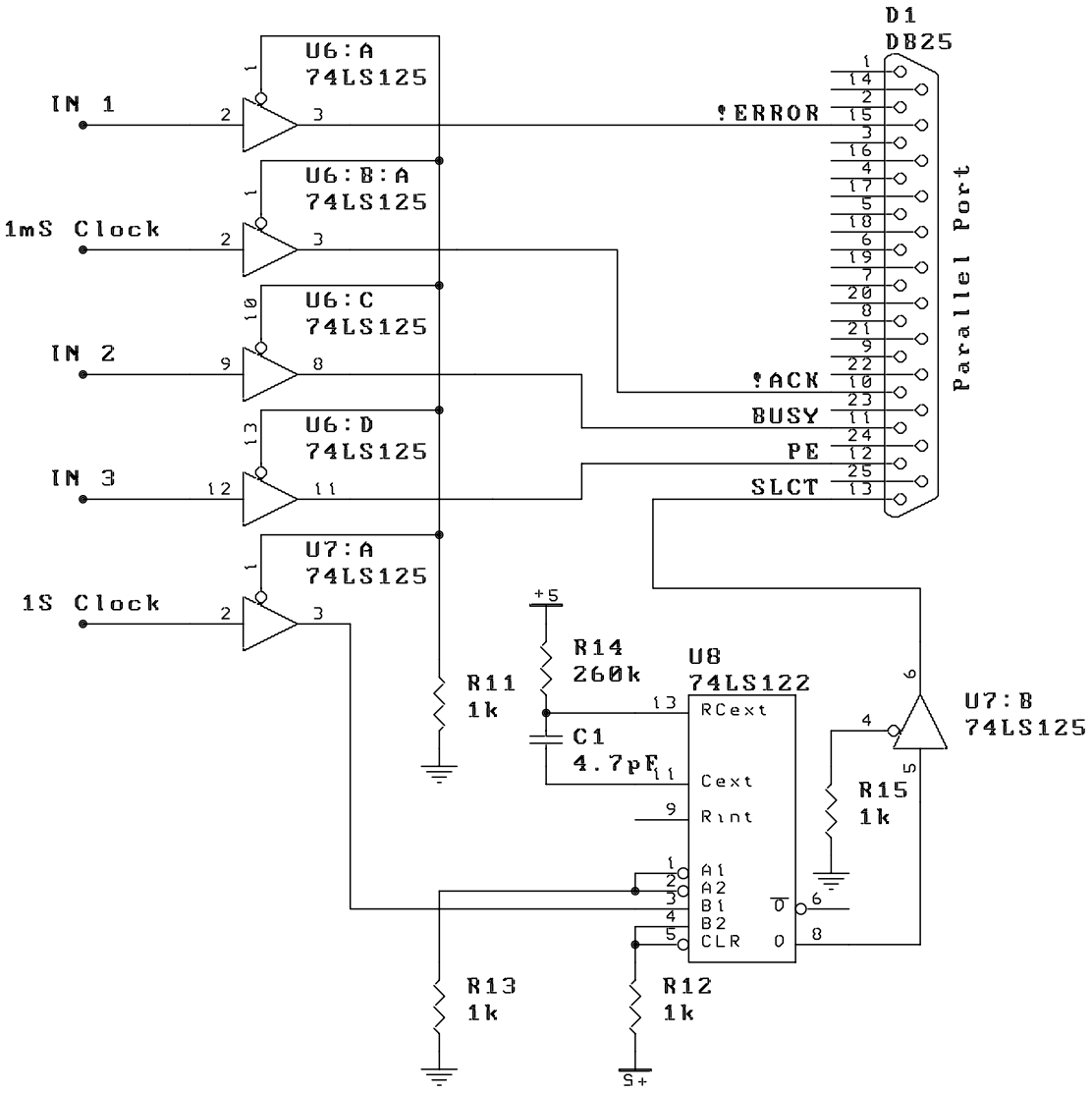

FIGURE 1. Schematic of the input electronics for the RT-Linux local clocks. Note that this does not show the bypass capacitors required for each chip.

TABLE 1. Input Connections

Input	Signal Name	Use
In 1	$\overline{\text{ERROR}}$	General purpose TTL input
In 2	BUSY	General purpose TTL input
In 3	PE	General purpose TTL input
1-mS Clock	$\overline{\text{ACK}}$	1-mS interrupts
1-S Clock	SELECT	1-S synchronization

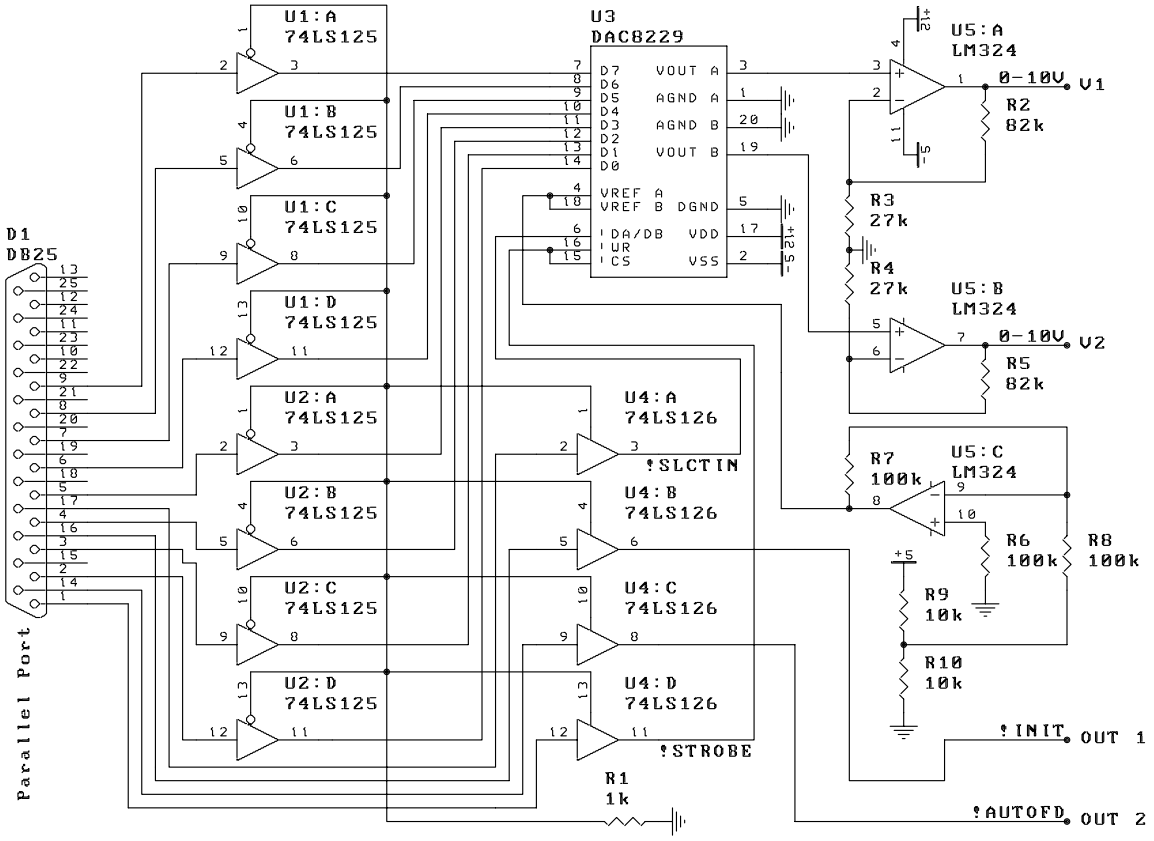

FIGURE 2. Schematic of the output electronics for the RT-Linux local clocks. Note that this does not show the bypass capacitors required for each chip.

1-second tick. Thus, when the interrupt has been fired, the interrupt service routine has 0.5 milliseconds to check if the 1-second clock tick has also gone by. See section 4 for more on the software clock tracking algorithm.

3.2. Output Electronics and Connections

A schematic diagram of the output electronics is shown in Figure 2 and the mapping of output signals to the standard parallel port data and signal names is given in Table 2.

As for the input electronics, all lines are fully buffered. The general purpose output TTL lines, Out 1 and Out 2, are connected directly to their respective buffered outputs. The remaining signals are used to control the dual DAC chip (DAC8229).

The DAC chip requires a reference voltage, in this case supplied by a resistor divider network (R9 and R10) and an inverting amplifier (U5:C). Together these elements provide a -2.5 volt reference voltage. The output voltage of each DAC is given by

$$V_{\text{out}} = -V_{\text{ref}} \times \frac{\text{DATA}}{256} \tag{1}$$

where DATA represents the number sent to the port data register. The DAC outputs will

TABLE 2. Output Connections

Output	Register Name	Use
V 1	Data, $\overline{\text{SLCTIN}}$ and $\overline{\text{STROBE}}$	Device controllable 0-10 volt DAC output
V 2	Data, $\overline{\text{SLCTIN}}$ and $\overline{\text{STROBE}}$	Device controllable 0-10 volt DAC output
Out 1	$\overline{\text{INIT}}$	General purpose TTL output
Out 2	$\overline{\text{AUTOFD}}$	General purpose TTL output

therefore range from 0 to about 2.5 volts. The final two operational amplifiers are used to bring this into the range 0 to 10 volts.

Two control lines are required to make the DAC8229 function correctly. One selects between the two DACs, while the second is used for latching the data. Table 3 shows the register values required to control the DAC operation. Remember that before any control operation some data must be present in the data register. In this table $\downarrow$ signifies a transition from 1 to 0. Also note that the register values listed are the *software* values placed in the control register not the TTL values present on the port output. The writing process therefore

TABLE 3. DAC Control Signals

SLCT IN	STROBE	DAC 1	DAC 2
1	1	Write	Hold
1	$\downarrow$	Latch	Hold
1	0	Hold	Hold
0	1	Hold	Write
0	$\downarrow$	Hold	Latch
0	0	Hold	Hold

consists of three operations: select the DAC you want to write to; send the data; and finally latch the data.

4. SOFTWARE

All control programs, except for special cases like the TCS and OPLE systems, will run under RT-Linux and follow the CHARA coding standards (see Technical Report TR70). Furthermore, a standard RT module, chara_sched, will be used to schedule real time tasks and act as an interface between the user interface software, the local clock described above, and the real-time tasks.

The CHARA scheduler is a simple interrupt driven real time scheduler fired by the 1-millisecond clock. When loaded, the module initializes the port and interrupts, creates communication channels between real-time tasks and the user interface program, initializes a command structure, sets the DACs and output TTL lines to zero and starts the local clock. The communications between real-time and asynchronous software is handled using RT-FIFOs as described in the RT-Linux documentation. Three FIFOs are provided, COMM_FIFO, DATA_FIFO and ERR_FIFO, which have obvious uses. The command and

error channels are handled by the chara scheduler for, strangely enough, processing commands and errors. Commands can be created or destroyed by real time tasks and consist of a single number (0–255), indicating the command, and a single datum (0–255). Errors are written to ERR_FIFO using a printf() like routine provided by chara_sched. The data channel is for use by the real time tasks and it is up to the programmer of each device to define the data formats.

While running, the first job of the scheduler is to increment the local time, and then poll to see if the 1-second tick has also fired in the last period. If it has, the local clock is checked to see if its value represents a whole second, and if not it is adjusted so that it does. The software will keep a record of skipped ticks for debugging purposes. It is also possible that the user interface program has previously sent a message containing the 'real' time of the next 1-second tick. In this case the scheduler, when it sees the next second tick, forces the local clock to the correct value. The local time is available to all tasks running on the system via a function (which, for the sake of performance, is in fact a macro).

The second job of the scheduler is to run the real time tasks at appropriate times. While usually only a single task is run in each device it will be possible to run several. The scheduler does no testing to check that the tasks run within their alloted time period, so it is up to each programmer to check this (for which they will find the spare output TTL ports and DACs very handy).

The scheduler also provides abstracted access to the DACs and the spare input and output TTL lines. These are in the form of macro functions, such as

```
out1_on();
dac1(value);
x = in1();
```

and so on. These are examples and subject to name changes. Consult the CHARA Scheduler manual (to be written) for the real details.

When unloaded the scheduler module releases the port interrupt and removes the communication fifos before exiting. It will also reset the DACs and output TTL ports. Remember it will not be possible to unload the scheduler while any modules exist that rely on it.