# PIC PULSE GENERATOR DOCUMENTATION (Version 1.11)

Last revised: 13–NOV–2006 20:06:26.
Note: New/altered material has been highlighted in RED.

(1) History & Introduction

The PIC Pulse Generator is the latest in a line of Pulse Generators designed and build by the NMR group. The first unit due to Tony Kent is a standalone design based on a Z80 microprocessor with control and display via front panel keys and 7–segment LED displays.

The next Pulse Generator design used a BBC B Microcomputer to enter, edit, display and generate the pulse sequence. Counter/timers in the machine's VIA 6522 I/O chip actually produced the raw pulses which were cleaned up and made available to the outside world by a simple user–port plug–in demultiplexer box. With almost all of the work being done in software it made for a flexible easy–to–modify pulse generator. The idea for the editing display is from Brian Cowan and the software was written by the author.

Upgrading to the then new 32bit RISC processor Archimedes machines, the next pulse generator (software) used the same demultiplexer box, which was driven again by a 6522 VIA chip on the plug–in Acorn I/O card which emulated the I/O facilities of the BBC Micro on the Archimedes. Unfortunately this design suffered from delay–length 'jitter' of the order of a few μS. The jitter is due to the refresh of the computer's DRAMS, making it difficult or impossible to eliminate. While not making the system unuseable it is undesirable. As a consequence this design was never used in experimental work and prompted yet another design.

The 50MHz pulse generator was designed with very short (sub–microsecond) pulses for NMR work on solids in mind. It uses the same Archimedes data entry/display system and is also controlled in realtime by the Archimedes via the I/O card's 1MHz bus. Both pulses and delays are produced by custom 32bit, mutually triggering counter/timers inside the 50MHz pulse generator box. The logic was designed by Alan Betts. This approach eliminates the delay–jitter problems of the previous design.

There are a number of drawbacks to these computer based pulse generators which have prompted the design of the PIC pulse generator. Some of these are;

- Locks up a expensive computer for a relatively trivial job which could easily be doing something more befitting its capabilities.
- Computers and their monitors are notorious sources of RF noise which finds its way into spectrometers etc. to which the pulse generator is attached. The noise stems from crystal clocks inside the computer which together with the Fourier components of their division frequencies (production of which are an inherent feature of the busses of microprocessors) produce a broad frequency swathe of radio frequency pollution. Opto–isolation is usually only partially effective in removing the noise which is often highly frequency stable making it difficult to remove by signal averaging. This is because it

can get into spectrometers via a variety of routes; Busses inside and outside the computer/monitor can act as antennas, radiating the signals. Being high frequencies they can capacitatively couple across opto–isolators. They can be induced in the braid of coax cables, propagate via earth loops etc.

The PIC pulse generator is based around 8+3 PIC16C84 or PIC16F84 microcontroller chips and a 2KByte EEPROM memory chip to store pulse sequences. Each PIC16C/F84 chip contains only 1Kword EEPROM program memory, 36/68 bytes of RAM, and 64 bytes of non–volatile data EEPROM – hence the need for the multiple chips across which the functionality of the design is split. PIC chips use the Harvard Architecture for security reasons and hence all their busses (to memory, I/O ports etc.) are internal to the chip which should eliminate the bus–radiating problems of computers and microprocessor designs using the traditional Von Neumann architecture. The instrument's display uses a 4×16 digit LCD module for data entry/display plus a few LEDs. Again this should be much less noisy than a computer monitor. The design attempts to reproduce the data entry & editing display of the previous computer based ones by using the LCD as a 'window' which scrolls up and down a 'virtual' editor display.

New Features in PIC Pulse Generator Version 1.11:
- This release is a bugfix to version 1.10 only. The bug symptom is premature shutdown of the slave pulse generator during its recycle time period when the user selects a slave recycle time < the master recycle time. Typically the slave ceases pulse generation during its first recycle period. The bugfix has necessitated adding another 7uS to the minimum permitted recycle time.

(2) User Guide
(A) Initial Data Entry.
This is the same as previous designs. The required input is;
- Number of Pulses in the sequence
- Pulse to recycle to
- Number of times to recycle (Enter 0 for infinity)
- The *Pn*/*Dn* labels and channel specifiers for the *N* pulses specified above. The *Pn*/*Dn* labels can be anything in the range P1–P8 & D1–D8. The channel specifiers list the channel(s) each pulse will appear on. They are lists of between 1 and 8 digits in the range 1–8 in any order. Egs. "5" for pulse#2 means the second pulse in the sequence will emerge on channel 5 only. "6254" for pulse#7 would mean the 7th pulse emerges on channels 2,4,5 & 6.
- The values and units of the pulses and delays. Values are in the range 1–16383 for pulses and *x*–16383 for delays, where the '*x*' depends on the position of the delay in the sequence. See specifications section below for more details. Units are one of; μS, mS, Seconds or Minutes. Values are typed first and entered

with the <Enter> key. Units are entered by hitting <Enter> over the the required symbol in the "μmSM" prompt.

The pulse sequence is then complete and control passes to the editor menu.

(B) Pulse Sequence Editing.

As mentioned above the pulse sequence editor is accessed by scrolling the LCD window around the virtual display using the 4 cursor keys and modifying the sequence using other key−strokes. Here is an annotated example pulse sequence from the virtual display.

```
(a0)        ┌──────────────────┐
            ↓                  ↑
(a1)     P1 2 3 4 5 6 7 8
(a2)      D1 2 3 4 5 6 78
(a3)       C C C C C C C C
(a4)       1 1 1 1 3 4 5 3
(a5)         2   2   6
(a6)             3   7
(a7)             4
(a8)             5
(a9)             6
(a10)            7
(a11)            8

(b0)      P1=1μS
(b1)      P2=12μS
(b2)      P3=123μS
(b3)      P4=1234μS
(b4)      P5=12345μS
(b5)      P6=140mS
(b6)      P7=100mS
(b7)      P8=2089mS

(c0)      D1=100μS
(c1)      D2=12345μS
(c2)      D3=1mS
(c3)      D4=10mS
(c4)      D5=100mS
(c5)      D6=1000mS
(c6)      D7=1Sec
(c7)      D8=10Min

(d0)      Recycle# 65123
```

The editing options for each line/group of lines are given below.

(a0)         Both the number of pulses in the sequence and the recycle−to position can be

|           | altered by clicking the <Enter> key over the arrow–head or arrow–tail moving left or right to the desired new position and clicking <Enter> and second time. Naturally invalid choices such as recycle-to pulse > number–of–pulses are rejected. |
|-----------|---|

(a1–a2)     *Pn* and *Dn* pulse/delay labels are modified by pressing a number key over the pulse/delay # to be modified. The number pressed defines the new *Pn*/*Dn* label.

(a3)     Pressing a number key over the '*C*' for a particular pulse will add that channel to the list of channels the pulse will emerge from. That list is displayed vertically over lines a4–a11.

(a4–a11)     Pressing the <Delete> key over a channel specifier for a particular pulse will remove that channel from the list of channels the pulse will emerge from.

(b0–b7)     Both the *Pn* pulse label values and units can be modified here.

     (a)     To modify the value – overtype it with the new value and hit <u><Enter> over any part of the number</u> to enter it. Hit <Delete> to discard any changes made. Note: to change for example, 1000 → 100, the string "1000" can be overtyped with "0100".

     (b)     To modify the units – Hit <Enter> over any part of the units string, eg. over the 'S', 'e' or 'c' of "Sec". The units chooser string "µmSM" then appears. Finally, hit <Enter> over a symbol for the desired new units.

(c0–c7)     Editing the *Dn* delay labels is the same as for *Pn* pulse labels. There are some additional issues here which relate to µS units delays. If too small a value is entered for a particular delay pointed to by a label an error message will be printed. The message will refer to a delay for which the value was too small. Note: This does not necessarily imply that delay was the only one for which the value was too small. It does not even necessarily mean that delay had the longest minimum acceptable value of the delays referred to by that label. After the error message the delay value will be unchanged and the user must leave it unchanged or choose a larger value.

A similar situation can occur when *Dn* units are modified. Eg. if a particular *Dn* was 5mS and the user modified the units to µS an error will occur because the pulse generator cannot generate a delay as short as 5µS. The units will remain unchanged and the user must first change the *Dn* value to something acceptably larger and then change the units to µS. Note: Under certain circumstances *Dn* label values with µS units can alter 'spontaneously' by a few 10s of µS when edits are made in lines a0–a2. This is an unfortunate side effect of the way pulse sequence information is stored in the PIC chips memory together with the very limited amount of memory available in these chips. A future

revision of the software may modify this behaviour. In the mean time the user should check *Dn* label values after any modifications in lines a0–a2.

There are some additional issues concerning the recycle delay in slave sequences the user should be aware of: The slave board recycle period synchronisation modification in vers. 0.21 makes the slave board recycle delay value entered by the user effectively redundant since it is now controlled by the master pulse generator – that is, the slave boards pulse sequences now recycle when the master boards signals them to. For compatibility with the filesystem and master sequences, the user still must enter a value for the slave board recycle delay. <u>It is the responsibility of the user to enter a suitably short value such than the total slave board pulse sequence length is shorter than or equal to that of the total master pulse sequence</u>. Note: If this condition is not fulfilled, the result should be that the slave board sequence will recycle eg. once for every-other master pulse sequence recycle. Entering a slave recycle delay of 150μS should be suitable for all pulse sequences and is therefore the recommended value. For simple pulse sequences smaller values may be used if a very short repetition time is needed, check with PIC Pulse Generator specs. below or reduce from 150μS by trial-and-error in the pulse sequence editor. Please advise the author of any problems related to slave recycle delays.

(d0)     The Recycle value gives the number of times the repeat part of the pulse sequence encompassed by line a0 will be repeated. As with *Pn/Dn* label values the number is typed with numeric keys and entered with the <Enter> key. Specifying 0 produces an infinite recycle count.

(C) Accessing the pulse sequence EEPROM based file-system.

The file-system is accessed via the <Menu> key and may be entered at the initial "Nos Pulses ? " prompt and at most places in the Editor Menu described above. The initial menu appears as follows;

```
Filesys Slv Brds
```
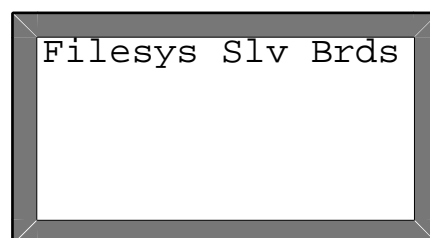
Scrolling left with the ← cursor key and hitting <Enter> over any part of the string "Filesys" will access the file-system menu which appears as follows, eg;

```
LD SV RN DL EXIT



12   Tom's T1
```

From here;

- The ← & → cursor keys can be used to chose between the options displayed on the top row (respectively LOAD, SAVE, RENAME, DELETE the current pulse sequence and EXIT the file-system menu.
- The ↑ & ↓ cursor keys scroll through the file-system's catalogue of pulse sequences. The example above shows the 12th pulse sequence whose name is "Tom's T1" is the currently selected one. In this version of the pulse generator 26 pulse sequences can be stored by the file-system.

- Loading a pulse sequence – Press <Enter> while the cursor is over the 'LD'. If successful the system responds with 'Done.'

- Saving a pulse sequence –  Press <Enter> while the cursor is over the 'SV'. The menu then displays, eg.;

```
LD SV RN DL EXIT


New?
12   Tom's T1
```

The pulse sequence can then be saved with the same name (Tom's T1) or a new name can be entered. Hit <Enter> to save under the same name. Hit ↑ or ↓ cursor keys to enter a new name. The menu then responds;
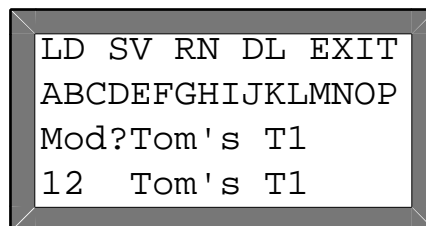
```
LD SV RN DL EXIT
ABCDEFGHIJKLMNOP
New?
12   Tom's T1
```

The new filename is constructed by choosing characters from line#2 and inserting them into line#3. The ↑ & ↓ keys toggle the cursor between lines 2 & 3. The ← & → keys select which character to choose from line#2 and where to place it in the filename being constructed in line#3. Hitting <Enter> over a character in line#2 selects that character and hitting <Enter> over a position in line#3 inserts the

selected character at that position. Hitting <Enter> over line#3 a second time completes the operation and saves the pulse sequence. Any‡ of the 256 characters generated by the display can be used in a filename and are accessed by using the ← and → keys to scroll line#2. Note: The display generates many special symbol characters as well as a variety of accented characters. Some example filenames/ pulse sequences have been pre-inserted into the EEPROM.

‡ There is one exception. The undefined character which occurs one place to the left of the first instance of the 'μ' may not be used.

- Renaming a pulse sequence –  Press <Enter> while the cursor is over the 'RN'. The menu then displays, eg.;

```
LD SV RN DL EXIT
ABCDEFGHIJKLMNOP
Mod?Tom's T1
12  Tom's T1
```

The procedure for modifying the filename displayed in line#3 with characters selected from line#2 is the same as described above for saving a pulse sequence with a new name.

- Deleting a pulse sequence –  Press <Enter> while the cursor is over the 'DL'. The menu then responds with "Sure?". Hit <Enter> to confirm the delete operation or hit any other key to back out.

- Exiting the file-system menu –  Press <Enter> while the cursor is over the 'EXIT'. If a pulse sequence was loaded from the EEPROM or a pulse sequence was already present before the file-system menu was invoked, control passes to the editor menu, otherwise control passes to the initial data entry procedure for a pulse sequence to be entered.

(D) Operating the Slave pulse generator boards.

The Slave pulse generator boards are accessed in exactly the same way as the Master pulse generator. Two new concepts and a new menu have been added to control the Slave boards. The two concepts are;

(1) The notion of a 'currently selected pulse generator board'
(2) The Enable/Disable status of a Slave pulse generator board.

Both entities can be displayed and altered via the Slave Board's menu which is accessed from the main menu;

```
Filesys Slv Brds
```

by hitting <Enter> over any part of the string `"Slv Brds"`. An example Slave boards menu is;

```
Mastr/Slave Menu
Status: EDD
Selected: 0
```

The Enable/Disable status of the 3 slave boards is shown on line 2 and the currently selected pulse generator board is shown on line 3. In the example slave board#1 is enabled, slave boards#2 & #3 are disabled and the currently selected pulse generator board is board#0 – the master pulse generator. The currently selected pulse generator board can be changed by hitting a numeric key in the range [0–3], 0 being the master pulse generator board and 1–3 being the 3 slave boards. The enable/disable status of the 3 slave boards can be changed by making that slave board the currently selected board (ie. hit a numeric key 1–3) and then using any of the cursor keys to toggle its enable/disable status. To leave the slave boards menu, hit <Enter> which returns the system to the main pulse sequence editing menu.

From this point hitting the <Start/Stop> key will cause the currently selected pulse generator board to be loaded with the pulse sequence displayed in the editor menu and pulse generation (on the master pulse generator board plus any enabled slave boards) to be started.

Alternatively, pressing the <Load> pulse sequence key will only result in the currently selected pulse generator board being loaded with the pulse sequence displayed in the editor menu. The <Load> pulse sequence key permits each of the pulse generator boards to be preloaded with what ever pulse sequence is required <u>before</u> starting pulse generation. This is the way a parallel pulse sequence is built up.

Note (1) As far as the editor menu and file-system is concerned, each pulse sequence loaded into each pulse generator board is a separate entity which must be saved as an entry in the file-system if it will need to be edited or reused in the future or on another pulse generator board.

Note (2) The master pulse generator board is always enabled and therefore <u>must</u> have a pulse sequence loaded into it for pulse generation to work. If the system is booted, pulse sequences only loaded into the slave board(s) and the <Start/Stop> key is pressed then pulse generation will not start. Instead, the error message `"MPS?"` (ie.

Master Pulse Sequence?) will be displayed and the pulse sequence editor menu will be reentered.

(E) Stopping/Starting Pulse Generation.

The <Start/Stop> key starts pulse generation. It will continue until the recycle count is reached or a user hits the key again to stop pulse generation. Note: If the <Start/Stop> key is pressed during a pulse that pulse will be cut short. For this reason it is a bad idea to hold down the <Start/Stop> key in the hope of 'manually recycling' the pulse sequence. The <Start/Stop> button's LED lights when pulse generation is in progress. After pulse generation has finished the editor display is reentered.

(F) Other Features.

- Pressing the <Escape> key software resets the pulse generator and restarts it at the initial pulse sequence entry point.
- The pulse generator has an external clock input facility. Selecting this will affect all timing within the system. This is because all timing within the unit is derived from the clock. Internally a 4MHz clock is used. If for example an 8MHz external clock is applied, all pulses and delays produced will be ½ the displayed values. A front panel LED indicates whether internal or external clock is in operation.
- The 'Bleep disable' key can be used to disable the bleeper which produces the 'keyclick' etc. This a latching key and its LED reflects the enable/disable status of the bleeper.
- The system has a hardware reset button on the back–panel. This can be used to regain control of the system in the event it hangs.
- In contrast with the previous microcomputer based pulse generators the PIC system does not use a relatively slow BASIC interpreter to produce the recycle delay. Consequently minimum recycle delays are only marginally longer than those of other delays. Be aware therefore that if you erroneously specify a recycle delay of say 100μS it will happily generate it, burning out the transmitter/frying your sample etc. You have been warned!

(G) The Computer Control Interface (CCI).

This section describes the operation of the pulse generator from a computer.

- The pulse generator's computer interface is an 8-bit parallel port with 2 handshake lines, modelled on the BBC Micro's User Port. Only the CB2 handshake line is currently used.
- The protocol used by the computer to communicate over this port with the pulse generator is the Multi Address System Protocol (MASP) due to Alan Betts (AKB). The rest of this section assumes familiarity with the MASP.
- Not all of the functionality of the pulse generator's front panel is

currently available over the CCI. The remote computer may define and download pulse sequences to the Master and Slave boards and also start pulse generation. There is no remote computer access to the pulse generator's EEPROM filesystem. Unlike the simpler MASP devices the pulse generator does not respond instantly (eg. typically within <100nS) to commands over the MASP bus. This is because the pulse generator's MASP parsing/handling runs on microcontrollers rather than dumb logic. It is essential the computer programmer takes account of these limitations when coding. Techniques for dealing with them are discussed in more detail later. See also the in-program comments in the BASIC example programs.

- Note: Some example programs are given below in BBC BASIC. References in the following text to the BASIC examples are typeset in green for clarity.


- **<u>Controlling The Pulse Generator From The Computer.</u>**

**(1) Initialisation.** After booting the pulse generator will be in the MASP bus *unaddressed* state and must be put in the *addressed* state before it will accept any other commands. As laid down in the MASP specification, each device on the MASP bus has a unique 'factory defined' 5-digit address. The PIC Pulse generators have addresses of the form "PPGxy" where x & y will typically be numbers eg. "01", "02" etc. for each pulse generator built/upgraded. See PROCaddress_device in BASIC example programs.

Note (a). Before the remote computer sends the 5-digit address, it must 'get the attention' of the pulse generator's MASP bus handling code. The recommended way to do this write a zero-byte several times on the MASP bus with *bit7=0*. This MASP *command mode* input will be ignored by other MASP devices but the pulse generator will see the MASP data-valid hi→lo transitions and start listening for its upcoming MASP address. See PROCsee_dv in the BASIC example programs.

Note (b). The pulse generator must receive its 5-digit address bytes contiguously with each byte being received within the pulse generator's input byte timeout period of ~65mS, for it to become MASP *addressed*. The scheme allows the pulse generator's internal hardware to return to scanning the front panel keyboard if it does not become *addressed* within the timeout periods – eg. if some other MASP device on the bus was being accessed.

**(2) Command Mode.** Once *addressed*, the pulse generator will accept commands in MASP *Command Mode (bit7=0)* and *Data*

*Mode (bit7=1).* The pulse generator has no device dependent *Instruction Mode* commands – all device dependent commands to the instrument are sent in MASP *Data Mode*. The three essential MASP *Instruction Mode* commands as described in the document `'MAS_REVIEW.DOC'` by AKB are implemented and are;

'@' – Unaddress the device. ie. clear the `ENINT` 'flag' and return to front panel control.

'<' – Prepare to reverse buffers and send data to computer (*Read Device Mode*, sets 'flag' `ENWR=1`).

'>' – Revert to Computer → Device data mode (sets 'flag' `ENWR=0`).

Anything else sent in *Instruction Mode* is ignored.

**(3) Data Mode.** All *Data Mode* commands must be terminated with `<CR>`, that is ASCII (13 + 128). See `PROCwrite_data_string` in the BASIC example programs. The *Data Mode* commands closely model the front panel commands. The full set of *Data Mode* commands is as follows;

```
Command             Argument Range          Command Description        Equiv. line in Sec.(B)
NP=<n>              1<=n<=8                 Number of pulses in sequence       a0
RT=<m>              1<=m<=n                 Recycle-to position                a0
PL<y>=<x>           1<=y<=n, 1<=x<=8        Pulse Label                        a1
DL<y>=<x>           1<=y<=n, 1<=x<=8        Delay Label                        a2
CH<y>=<string>      eg. "1235678",1<=y<=n   Channel specification              a4-a11
PV<x>=<numeric string><units string>  Pulse Value                            b0-b7
                    1<=x<=8, <units string> is one of "uS", "mS",
                    "Sec" or "Min". NB: <numeric string> is
                    between 1 and 5 characters long,
                    eg. "PV6=12345uS".
DV<x>=<numeric string><units string>  Delay Value (args as Pulse value)      c0-c7
RC=<numeric string>                    Recycle count, range is [0-65535]      d0
                                           NB: 0=Infinite count
SB=<a>              0<=a<=3             Select pulse generator board,
                                           0=master, [1-3]=slave boards #[1-3]
BS='E' or 'D'                          Enable or disable selected board
LB                                     Load sequence into selected board
GO                                     Start pulse generation using pulse sequence
                                           created by remote computer.
GOI                                    Start pulse generation (immediate) - that is
                                           using whatever pulse sequence was previously
                                           loaded; from remote computer or via the front
                                           panel. Caution: No checks on pulse sequence
                                           validity are made - it is used 'as is'.
```

Miscellaneous notes: Commands `NP` through to `RC` are mandatory and must all be present to define a valid pulse sequence. As with front panel entry, by default the master board is selected and the three slave boards are disabled. If slave pulse boards are to be used, they must be Selected (SB=<a>), Enabled

(BS=E) and then Loaded (LB).

**(4) Read Device Mode.** In this special mode, described in detail in the document `'MAS_REVIEW.DOC'`, after receiving the `'<'` in *Command Mode*, the pulse generator is able to send strings back to the computer. In these responses, which can and should ideally be requested after every command is sent, the pulse generator indicates if the command was valid and if not, gives an indication of what was wrong with it. Most responses consist of two-letter strings. See `PROCdisp_str` and `FNread_data_string` in the BASIC example programs. All strings are <NULL> terminated. The complete list of responses follows;

| Response \| Meaning | Commands which may produce it | Explanation |
|---|---|---|
| ?? Default Response | ** Pulse generator response requested before any commands sent | |
| OK Valid Command | ** Command was OK ** | |
| UC Unrecognised Command | ** Command was unrecognised ** | |
| IA Invalid Argument | NP=<n> | Invalid <n> |
| | RT=<m> | Invalid <m> |
| | PL<y>=<x> | Invalid <x> or <y> |
| | DL<y>=<x> | Invalid <x> or <y> |
| | CH<y>=<string> | Invalid <y> or <string> |
| | PV<x>=<numeric string><units string> | Invalid <x> or <1 or >5 chars. in numeric string |
| | DV<x>=<numeric string><units string> | Invalid <x> or <1 or >5 chars in numeric string |
| | RC=<numeric string> | Nonnumeric chars in string |
| | BS=<arg> | <arg> is not 'D' or 'E' |
| IU Invalid Units | PV<x>=<numeric string><units string> | 1st char of units invalid |
| | DV<x>=<numeric string><units string> | 1st char of units invalid |
| TS Too Small | PV<x>=<numeric string><units string> | Number was too small |
| | DV<x>=<numeric string><units string> | Number was too small |
| TB Too Big | PV<x>=<numeric string><units string> | Number was too big |
| | DV<x>=<numeric string><units string> | Number was too big |
| MP Master Pulse? | GO | Pulse generation was requested but no pulse sequence was loaded into master pulse generator. See also front panel error "MPS?" |

**C**onsistency **C**hecking errors produced by GO and LB commands

| | |
|---|---|
| CCNP? | Number of Pulses (NP) was not in the range 1–8 |
| CCRT? | Recycle-to position (RT) was not in the range 1-<Number of Pulses> |
| CCPL<y>? | The label for the yth pulse (PL<y>=...) is missing |
| CCDL<y>? | The label for the yth delay (DL<y>=...) is missing |
| CCPV<x>? | The value+units of the xth pulse label (PV<x>=...) is missing |
| CCDV<x>? | The value+units of the xth delay label (DV<x>=...) is missing |

Consistency Checking on the pulse sequence is performed prior to pulse loading and pulse generation (not GOI) to ensure all the information needed to define the sequence is present and that it is internally consistent. If the LB or GO commands produce an

error, pulse sequence loading or pulse generation does not take place. The error should then be corrected before re-issuing the command.

Note: Occasionally, the computer may erroneously receive some '<' characters embedded in the response string from the pulse generator. This is caused by buffer direction switching mis-synchronisation between the computer (sending the '<' character to tell the pulse generator to switch to *read device mode*) and pulse generator. It is a benign bug but one which the computer programmer should be aware of if, for example, parsing the response string to look for results other than 'OK'.

**(5) Timing & Related Issues.** As mentioned above the pulse generator, cannot always be scanning its MASP bus port. Neither does it respond to commands on the MASP bus as fast as plain logic devices do. Also, the MASP protocol has no provision for a *busy-line* for the client device to signal the controlling computer that it is not ready for input.

In the example BASIC programs the general tardiness of the pulse generator in handling bus commands is dealt with by inserting some delays. 10mS delays have been inserted into the Data-Valid signalling, which is applied to each byte sent and received, to ensure it is slow enough to be seen by the pulse generator. Note: The 10mS is an arbitrary value, used in testing/ software development and significantly shorter values may well work just as well. See `PROCshort_delay` in the BASIC example programs.

After booting and also after pulse generation has ceased, the computer must 'get the attention' of the pulse generator's MASP bus handling code, before sending any other commands. As already mentioned, sending multiple <NULL> bytes on the MASP bus will cause the pulse generator to 'see' the Data-Valid line being toggled and will (re-)activate its MASP handling code ready to accept commands. See `PROCsee_dv` in the BASIC example programs. After pulse generation has finished, the front panel editor display will be updated with the just-executed pulse sequence and front panel commands can be entered again − eg. to examine the sequence in the LCD's editor display. It is probably inadvisable to actually alter the pulse sequence here because the computer interface is still active ( 'flag' `ENINT`=1) and subsequent commands from the computer will overwrite any changes made.

In the example BASIC programs, between each complete command (eg. `"NP=5"`) a delay of typically 2 seconds has been inserted. This arbitrary value was largely to permit

viewing of commands as they executed, during software development/debugging. It is likely that much shorter values will work too. See `PROCpause` in `PROCwrite_data_string` in the BASIC example programs.

It can be inconvenient for the computer to have to calculate the length of time a pulse sequence should take to execute and then wait that length of time before calling `PROCdisp_str` in order to check on the validity of the preceding `GO` command. See the BASIC example program ppg_demo which operates this way. A more elaborate version of the pulse generator response-reading code is presented in example program `ppg_demo2`. In this second example program `PROCdisp_str` is called immediately after the `GO` command is executed and `FNread_data_string` loops to sample the MASP bus until pulse generation has finished and the pulse generator is responding to MASP bus commands again. See `FNread_data_string` which is called by `PROCdisp_str` in ppg_demo and ppg_demo2 for the details of these bus reading schemes.

**(6) Front Panel LED indicators.** There are 2 front panel LEDs which relate to the operation of the CCI. The 'COMP' LED is lit when the pulse generator is under computer control – that is when it is in the MASP *addressed* state and the 'flag' `ENINT=1` as described in the document `'MAS_REVIEW.DOC'`. The 'DATA' LED gives a visual indication of data passing between the computer and pulse generator and is modelled on the data LEDs typically found on Ethernet cards and hubs etc. Note: At the time of writing, the current pulse generator boxes do not have a DATA LED – this should be added when those units are given this software (& hardware – see AKB) upgrade.

**(7) The Interactive MASP STOP Command.** The Interactive Stop Command for the MASP computer interface was introduced in Version 1.10. Previously, once pulse generation had been started by the MASP `GO` command the computer program had to wait until the pulse generator had finished executing the programmed sequence — for sequences with long delay times and/or large recycle counts this can be a considerable length of time. The Interactive Stop command provides the same feature as the Stop/Start front panel button — that is, pressing the button will halt pulse generation immediately. Now the computer can interactively stop pulse generation by writing the (lower case) '**s**' character on the MAS bus long enough for the PPG to *see* it. A

period of a second should be sufficient. No accompanying MASP Data Valid Hi→Lo transition is needed. Strictly speaking this goes outside the MASP. This is not a problem for two reasons;

(1) The PPG will be the MASP currently 'addressed' device on the bus so no other device on the same bus should respond even if it implements an '**s**' command.

(2) The PPG's instruction set is all upper-case (MASP data mode) and the '**s**' is lower-case so the PPG will not subsequently misinterpret the 's' as part of a true command.

**(8)    The BASIC Example Programs.** The two following programs are demonstrations of working codes which download pulse sequences to the pulse generator and then execute them. They are intended as a guide to programmers working in BBC BASIC/RISCOS or any other programming language/platform such as Labview/Microsoft and have been heavily commented to that end. The main part of the program at the top defines and downloads the pulse sequence, with the work implementing the MASP driver code etc. being done by lower level FuNctions and PROCedures. The first example `PPG_DEMO` produces a very simple pulse sequence to the master pulse board, and waits 'manually' while it executes. The second program `PPG_DEMO2` produces plus loads a couple of simple pulse sequences for two of the slave boards. Then it creates a more complicated sequence for the master board before giving the `GO` command to synchronously start the 3 boards generating their pulse sequences. `PPG_DEMO2` then calls `PROCdisp_str` to 'automatically' wait for pulse generation to finish and the response string to be returned by the pulse generator. Both programs end by giving the '@' command to *unaddress* the pulse generator and therefore relinquish control of the MASP bus.

Example Program PPG DEMO

```
REM > ppg_demo
REM This program runs under RISCOS and drives the PIC pulse generator
REM using the MAS protocol through the User Port. The User Port H/W is provided
REM by the 6522 VIA chip on the Acorn I/O Podule. The software interface is via
REM the BASIC 'SYS "Os_Byte" 150 and 151' calls. For details see the Acorn I/O
REM Podule guide and Rockwell data book.
REM T.Crane, 3-JUL-2000 21:06:31
REM Revised, 9-NOV-2000 16:22:18
PROCinit                                :REM Initialise the system
ON ERROR PROCerror
PROCsee_dv:                             REM Get attention of PPG MASP code
PROCaddress_device("PPG00"):            REM Send its unique address to PPG

REM Send Data Mode commands to the PPG and read back and display the
REM response to each one with PROCdisp_str
PROCdisp_str
```

```
PROCwrite_data_string("NP=1"):          PROCdisp_str
PROCwrite_data_string("RT=1"):          PROCdisp_str
PROCwrite_data_string("PL1=1"):         PROCdisp_str
PROCwrite_data_string("DL1=1"):         PROCdisp_str
PROCwrite_data_string("CH1=12345678"):  PROCdisp_str:REM put pulse#1 on all channels
PROCwrite_data_string("PV1=1000uS"):    PROCdisp_str
PROCwrite_data_string("DV1=1000uS"):    PROCdisp_str
PROCwrite_data_string("RC=1000"):       PROCdisp_str
PROCwrite_data_string("GO"):
REM Wait 2 seconds [1000*(1000uS pulse +1000uS delay] for pulses to finish
REM This is because PPG cannot communicate with us during pulse generation
TIME=0
REPEAT UNTIL (TIME > 200)
PROCsee_dv:                             REM Re-get attention of PPG MASP code
PROCdisp_str:                           REM Display result of GO command
REM Unaddress the PIC device and exit
PROCwrite_byte(ASC("@"))
REM PPG is now under front panel control again
END
:
DEFPROCinit
PRINT "Initialising bus..."
SYS "OS_Byte",151,&6C,244        :REM Initialise CB2=HI
PROCset_bus_read: REM Set MASP bus direction to read
ENDPROC
:
DEFPROCwrite_byte(byte%)
REM Write a byte on the MASP bus and send data valid
REM First check if bus is set to READ or WRITE
REM If MASP bus direction is READ then it must be set to WRITE
IF (rw_flag%<>0) THEN PROCset_bus_write
SYS "OS_Byte",151,&60,byte%
PROCdv
ENDPROC
:
DEFPROCset_bus_read
REM Set the MASP bus direction (ie. VIA data direction register) to READ
SYS "OS_Byte",151,&62,%00000000 : REM SET VIA DDRB TO READ MODE
rw_flag%=1 : REM Reflect bus direction in S/W flag
PRINT "******MASP bus set for READ"
ENDPROC
:
DEFPROCset_bus_write
REM Set the MASP bus direction (ie. VIA data direction register) to WRITE
SYS "OS_Byte",151,&62,%11111111 : REM SET VIA DDRB TO WRITE MODE
rw_flag%=0 : REM Reflect bus direction in S/W flag
PRINT "******MASP bus set for WRITE"
ENDPROC
:
DEFPROCdv
REM Send Data Valid (Hi->Lo on the CB2 line) and wait for bus to "settle"
SYS "OS_Byte",151,&6C,244 : REM CB2=HI
PROCshort_delay
SYS "OS_Byte",151,&6C,192 : REM CB2=LO
PROCshort_delay
SYS "OS_Byte",151,&6C,244 : REM CB2=HI
PROCshort_delay
ENDPROC
:
DEFPROCshort_delay
REM A short delay for the slow PIC H/W to see DV signals/settling time etc.
REM Settling time is 10mS
TIME=0
```

```
REPEAT
UNTILTIME>=1 : REM WAS 10
ENDPROC
:
DEFPROCwrite_data_string(string$)
REM This subroutine writes a data string on the MASP bus
REM Note: DATA as opposed to INSTRUCTIONS must have bit7 set
PRINT "Preparing to send data string to device... """;
FOR A%=1 TO LEN(string$)
 VDU ASC(MID$(string$,A%,1))
 PROCwrite_byte(128+ASC(MID$(string$,A%,1)))
NEXT
PRINT """"'"Sending string terminator <CR>"
PROCwrite_byte(13+128): REM <CR> Terminates string
PROCpause
PRINT "Data string now sent"
ENDPROC
:
DEFPROCaddress_device(string$)
REM Send the 5 bytes of chars and numbers to address the device
PRINT "Preparing to address device ";string$;"..."
FOR A%=1 TO 5
 PROCwrite_byte(ASC(MID$(string$,A%,1)))
NEXT
PRINT "Device ";string$;" now addressed"
ENDPROC
:
DEFPROCpause
REM Simple subroutine to wait a 2 seconds for the user
REM to read the display etc.
TIME=0
REPEAT
UNTILTIME>200
ENDPROC
:
DEFFNread_data_string
REM Simple version - must not be called during pulse generation
LOCAL string$
string$=""
REM This function reads a data string from the device
PRINT "Preparing to read a string from the PIC device..."
PROCwrite_byte(ASC("<"))         :REM Send MASP "<" char and Hi->Lo->Hi on DV
PROCset_bus_read                 :REM Set MASP bus direction to read
REPEAT
 SYS "OS_Byte",151,&6C,192       :REM Send CB2=LO
 PROCshort_delay                 :REM Wait for PIC to put byte on bus
 SYS "OS_Byte",150,&60 TO ,,A%   :REM Read byte put on bus by PIC device
 REM Read data should have the top bit set
 IF A% < 128 THEN PRINT "ERROR; A%=";A% : STOP
 A% = A% AND 127                 :REM Strip the top bit
 IF (A% >=32 AND A% < 127) THEN
  string$=string$+CHR$A%
 ENDIF
 PROCshort_delay                 :REM Wait for PIC to start watching for Lo->Hi on DV
 SYS "OS_Byte",151,&6C,244       :REM Send CB2=HI
 PROCshort_delay                 :REM Wait for PIC device to disable its O/P buffer
UNTIL (A% = 0)
REM Terminator seen, all bytes now read from PIC device, Now;
REM (1) Turn bus direction to write
REM (2) Put ">" char on bus to signal PIC device must exit "Read Device Mode"
REM (3) Send DV (Hi->lo) so PIC device receives ">" & clears its ENWR flag
PROCwrite_byte(ASC(">"))
=string$
```

```
ENDPROC
:
DEFPROCdisp_str
REM Read and display the error_string from the PPG
readstr$=FNread_data_string            :REM Read a data string from device
PRINT "String was """;readstr$;""""
ENDPROC
:
DEFPROCerror
Clean up MASP bus after any BASIC interpreter errors
PRINT '"ERROR HANDLER: Resetting CB2=HI (ie. DV=HI)"
SYS "OS_Byte",151,&6C,244        :REM Reset CB2=HI
STOP
ENDPROC
:
DEFPROCsee_dv
REM Send multiple dummy data-valids to make the PPG invoke its MASP
REM handling code and prepare to read from the MASP bus
FOR I%=1 TO 10
 PROCwrite_byte(0): REM Make it see dv
NEXT I%
ENDPROC
```

## Example Program PPG DEMO2

```
REM > ppg_demo2
REM This program runs under RISCOS and drives the PIC pulse generator
REM using the MAS protocol through the User Port. The User Port H/W is provided
REM by the 6522 VIA chip on the Acorn I/O Podule. The software interface is via
REM the BASIC 'SYS "Os_Byte" 150 and 151' calls. For details see the Acorn I/O
REM Podule guide and Rockwell data book.
REM T.Crane, 3-JUL-2000 21:06:31
REM Revised, 9-NOV-2000 20:48:18
PROCinit                                 :REM Initialise the system
ON ERROR PROCerror
REM Make the PPG see dv and prepare to read from the MASP bus
PROCsee_dv:                              REM Get attention of PPG MASP code
PROCaddress_device("PPG00")             :REM Send its unique address to PPG

REM A more complicated example - First a couple of simple pulse sequences
REM are sent to slave boards 1 & 2. Then an 8-pulse sequence is sent to
REM the Master board.
REM Send Data Mode commands to the PPG and read back and display the
REM response to each one with PROCdisp_str
REM First the Slave boards pulse sequences
REM A 1-pulse sequence for Slave board#1
PROCwrite_data_string("NP=1"):          PROCdisp_str
PROCwrite_data_string("RT=1"):          PROCdisp_str
PROCwrite_data_string("PL1=1"):         PROCdisp_str
PROCwrite_data_string("DL1=1"):         PROCdisp_str
PROCwrite_data_string("CH1=1"):         PROCdisp_str: REM Put it on Channel 1 only
PROCwrite_data_string("PV1=1mS"):       PROCdisp_str
PROCwrite_data_string("DV1=100mS"):     PROCdisp_str: REM 100mS arbitrary (<2Sec)
PROCwrite_data_string("RC=0"):          PROCdisp_str: REM RC=0 is OK on Slave boards
PROCwrite_data_string("SB=1"):          PROCdisp_str: REM select Slave board#1
PROCwrite_data_string("BS=E"):          PROCdisp_str: REM Enable it
PROCwrite_data_string("LB"):            PROCdisp_str: REM Load its pulse sequence
REM ─────────────────────────────────────────
REM A 2-pulse sequence for Slave board#2
PROCwrite_data_string("NP=2"):          PROCdisp_str
PROCwrite_data_string("PL2=6"):         PROCdisp_str
```

```
PROCwrite_data_string("DL2=6"):          PROCdisp_str
PROCwrite_data_string("CH2=12"):         PROCdisp_str: REM Put pulse#2 on channels 1&2
PROCwrite_data_string("PV6=10mS"):       PROCdisp_str
PROCwrite_data_string("DV1=20mS"):       PROCdisp_str
PROCwrite_data_string("DV6=100mS"):      PROCdisp_str: REM 100mS arbitrary (<2Sec)
PROCwrite_data_string("SB=2"):           PROCdisp_str: REM select Slave board#2
PROCwrite_data_string("BS=E"):           PROCdisp_str: REM Enable it
PROCwrite_data_string("LB"):             PROCdisp_str: REM Load its pulse sequence
REM ——————————————————————————————————————
REM Now the Master board pulse sequence
PROCwrite_data_string("NP=8"):           PROCdisp_str
PROCwrite_data_string("RT=2"):           PROCdisp_str
PROCwrite_data_string("PL1=1"):          PROCdisp_str
PROCwrite_data_string("PL2=2"):          PROCdisp_str
PROCwrite_data_string("PL3=3"):          PROCdisp_str
PROCwrite_data_string("PL4=4"):          PROCdisp_str
PROCwrite_data_string("PL5=5"):          PROCdisp_str
PROCwrite_data_string("PL6=6"):          PROCdisp_str
PROCwrite_data_string("PL7=7"):          PROCdisp_str
PROCwrite_data_string("PL8=8"):          PROCdisp_str
PROCwrite_data_string("DL1=1"):          PROCdisp_str
PROCwrite_data_string("DL2=2"):          PROCdisp_str
PROCwrite_data_string("DL3=3"):          PROCdisp_str
PROCwrite_data_string("DL4=4"):          PROCdisp_str
PROCwrite_data_string("DL5=5"):          PROCdisp_str
PROCwrite_data_string("DL6=6"):          PROCdisp_str
PROCwrite_data_string("DL7=7"):          PROCdisp_str
PROCwrite_data_string("DL8=8"):          PROCdisp_str
PROCwrite_data_string("CH1=12345678"):   PROCdisp_str: REM Put pulse#1 on all channels
PROCwrite_data_string("CH2=12345678"):   PROCdisp_str: REM Put pulse#2 on all channels
PROCwrite_data_string("CH3=12345678"):   PROCdisp_str: REM Put pulse#3 on all channels
PROCwrite_data_string("CH4=12345678"):   PROCdisp_str: REM Put pulse#4 on all channels
PROCwrite_data_string("CH5=12345678"):   PROCdisp_str: REM Put pulse#5 on all channels
PROCwrite_data_string("CH6=12345678"):   PROCdisp_str: REM Put pulse#6 on all channels
PROCwrite_data_string("CH7=12345678"):   PROCdisp_str: REM Put pulse#7 on all channels
PROCwrite_data_string("CH8=12345678"):   PROCdisp_str: REM Put pulse#8 on all channels
PROCwrite_data_string("PV1=10uS"):       PROCdisp_str
PROCwrite_data_string("PV2=20uS"):       PROCdisp_str
PROCwrite_data_string("PV3=30uS"):       PROCdisp_str
PROCwrite_data_string("PV4=40uS"):       PROCdisp_str
PROCwrite_data_string("PV5=50uS"):       PROCdisp_str
PROCwrite_data_string("PV6=60uS"):       PROCdisp_str
PROCwrite_data_string("PV7=70uS"):       PROCdisp_str
PROCwrite_data_string("PV8=80uS"):       PROCdisp_str
PROCwrite_data_string("DV1=200uS"):      PROCdisp_str
PROCwrite_data_string("DV2=210uS"):      PROCdisp_str
PROCwrite_data_string("DV3=220uS"):      PROCdisp_str
PROCwrite_data_string("DV4=230uS"):      PROCdisp_str
PROCwrite_data_string("DV5=240uS"):      PROCdisp_str
PROCwrite_data_string("DV6=250uS"):      PROCdisp_str
PROCwrite_data_string("DV7=260uS"):      PROCdisp_str
PROCwrite_data_string("DV8=2Sec"):       PROCdisp_str
PROCwrite_data_string("RC=60"):          PROCdisp_str
PROCwrite_data_string("SB=0"):           PROCdisp_str: REM Reselect Master board
PROCwrite_data_string("GO")
REM Call PROCdisp_str immediately to display result of GO command and let it
REM wait until pulse generation finishes and PPG will talk to us again.
REM Note: A call to PROCsee_dv is not needed here because the outer
REM       REPEAT-UNTIL loop in FNread_data_string has the side-effect of
REM       toggling the Data-Valid line to get the attention of the MASP
REM       handling code.
PROCdisp_str
REM Unaddress the PIC device and exit
```

```
PROCwrite_byte(ASC("@"))
END
:
DEFPROCinit
PRINT "Initialising bus..."
msg%=1 :REM Initialise to 'Messages On'
SYS "OS_Byte",151,&6C,244        :REM Initialise CB2=HI
PROCset_bus_read: REM Set MASP bus direction to read
ENDPROC
:
DEFPROCwrite_byte(byte%)
REM Write a byte on the MASP bus and send data valid
REM First check if bus is set to READ or WRITE
REM If MASP bus direction is READ then it must be set to WRITE
IF (rw_flag%<>0) THEN PROCset_bus_write
SYS "OS_Byte",151,&60,byte%
PROCdv
ENDPROC
:
DEFPROCset_bus_read
REM Set the MASP bus direction (ie. VIA data direction register) to READ
REM msg%=1 indicates if message below should be printed
SYS "OS_Byte",151,&62,%00000000 : REM SET VIA DDRB TO READ MODE
rw_flag%=1 : REM Reflect bus direction in S/W flag
IF msg% THEN PRINT "******MASP bus set for READ"
ENDPROC
:
DEFPROCset_bus_write
REM Set the MASP bus direction (ie. VIA data direction register) to WRITE
REM msg%=1 indicates if message below should be printed
SYS "OS_Byte",151,&62,%11111111 : REM SET VIA DDRB TO WRITE MODE
rw_flag%=0 : REM Reflect bus direction in S/W flag
IF msg% THEN PRINT "******MASP bus set for WRITE"
ENDPROC
:
DEFPROCdv
REM Send Data Valid (Hi->Lo on the CB2 line) and wait for bus to "settle"
SYS "OS_Byte",151,&6C,244 : REM CB2=HI
PROCshort_delay
SYS "OS_Byte",151,&6C,192 : REM CB2=LO
PROCshort_delay
SYS "OS_Byte",151,&6C,244 : REM CB2=HI
PROCshort_delay
ENDPROC
:
DEFPROCshort_delay
REM A short delay for the slow PIC H/W to see DV signals/settling time etc.
REM Settling time is 10mS
TIME=0
REPEAT
UNTILTIME>=1 : REM WAS 10
ENDPROC
:
DEFPROCwrite_data_string(string$)
REM This subroutine writes a data string on the MASP bus
REM Note: DATA as opposed to INSTRUCTIONS must have bit7 set
PRINT "Preparing to send data string to device... """;
FOR A%=1 TO LEN(string$)
 VDU ASC(MID$(string$,A%,1))
 PROCwrite_byte(128+ASC(MID$(string$,A%,1)))
NEXT
PRINT """"'"Sending string terminator <CR>"
PROCwrite_byte(13+128): REM <CR> Terminates string
```

```
      PROCpause
      PRINT "Data string now sent"
      ENDPROC
      :
      DEFPROCaddress_device(string$)
      REM Send the 5 bytes of chars and numbers to address the device
      PRINT "Preparing to address device ";string$;"..."
      FOR A%=1 TO 5
       PROCwrite_byte(ASC(MID$(string$,A%,1)))
      NEXT
      PRINT "Device ";string$;" now addressed"
      ENDPROC
      :
      DEFPROCpause
      REM Simple subroutine to wait a 1 seconds for the user to read the display etc.
      TIME=0
      REPEAT
      UNTILTIME>100
      ENDPROC
      :
      DEFFNread_data_string
      REM More sophisticated version - It can be called during pulse generation.
      REM It works by constantly attempting to read a byte from the PPG on the
      REM MASP bus in the outer REPEAT-UNTIL loop. If the pulse generator is
      REM producing pulses and therefore not listening on the MASP bus, its port
      REM lines will float high and 0xFF will always be read. When pulse
      REM generation ceases, the inner REPEAT-UNTIL loop will be able to read
      REM the characters forming the response to the last command and construct
      REM the string it returns in variable 'string$'.
      LOCAL string$
      string$=""
      REM This function reads a data string from the device
      PRINT "Preparing to read a string from the PIC device..."
      ff_count%=0
      msg%=0 :REM Turn Messages Off while reading / waiting to read string
      REM Check if &FF was read - if so loop until PPG is listening again
      REPEAT
       PROCwrite_byte(ASC("<"))          :REM Send MASP "<" char and Hi->Lo->Hi on DV
       PROCset_bus_read                  :REM Set MASP bus direction to read
       REPEAT
        SYS "OS_Byte",151,&6C,192        :REM Send CB2=LO
        PROCshort_delay                  :REM Wait for PIC to put byte on bus
        SYS "OS_Byte",150,&60 TO ,,A%    :REM Read byte put on bus by PIC device
        REM Read data should have the top bit set
        IF A% < 128 THEN PRINT "ERROR; A%=";A% : STOP
        REM First check if &FF was received (ie. PPG not listening)
        IF (A%=&FF) THEN
         ff_count%+=1 :REM Read &FF so increment ff_count%
        ELSE
         ff_count%=0  :REM valid char so zero ff_count%
        ENDIF
        A% = A% AND 127                  :REM Strip the top bit
        IF (A% >=32 AND A% < 127) THEN
         string$=string$+CHR$A%
        ELSE
        ENDIF
        PROCshort_delay                  :REM Wait for PIC to start watching for Lo-> Hi on DV
        SYS "OS_Byte",151,&6C,244        :REM Send CB2=HI
        PROCshort_delay                  :REM Wait for PIC device to disable its O/P buffer
       UNTIL (A% = 0 OR ff_count% <> 0)
      UNTIL (ff_count% = 0)
      msg%=1 :REM Turn Messages on again
      REM Null terminator seen, all bytes now read from PIC device, Now;
```

```
REM (1) Turn bus direction to write
REM (2) Put ">" char on bus to signal PIC device must exit "Read Device Mode"
REM (3) Send DV (Hi->lo) so PIC device receives ">" & clears its ENWR flag
PROCwrite_byte(ASC(">"))
=string$
ENDPROC
:
DEFPROCdisp_str
REM Read and display the error_string from the PPG
readstr$=FNread_data_string          :REM Read a data string from device
IF readstr$ <> "OK" AND readstr$ <> "??" THEN
 VDU7
 PRINT "Caution: String was """;readstr$;""""`
 dummy%=INKEY(400)
ELSE
 PRINT "String was """;readstr$;""""
ENDIF
ENDPROC
:
DEFPROCerror
PRINT '""ERROR HANDLER: Reseting CB2=HI (ie. DV=HI)"
SYS "OS_Byte",151,&6C,244      :REM Reset CB2=HI
REPORT:PRINT " at line "ERL
STOP
ENDPROC
:
DEFPROCsee_dv
REM Make the PPG see dv and prepare to read from the MASP bus
FOR I%=1 TO 10
 PROCwrite_byte(0): REM Make it see dv
NEXT I%
ENDPROC
```

(3) Specifications (individual pulse board):

| | |
|---|---|
| Number of pulses generated: | $1 \leqslant n \leqslant 8$. |
| Number of possible recycle–to positions: | $1 \leqslant m \leqslant 8$, $m \leqslant n$. |
| Number of independently specifiable pulse lengths: | 1–8. |
| Number of independently specifiable delay lengths: | 1–8. |
| Number of channels from which any pulse may emerge: | 1–8. |
| Minimum & maximum pulse lengths (4MHz clock): | 1–16383μS,  1μS steps |
| | 1–16383mS,  1mS steps |
| | 1–16383Sec, 1Sec steps |
| | 1–16383Min, 1Min steps |
| Minimum & maximum delay lengths (4MHz clock): | $x$–16383μS,  1μS steps |
| | 1–16383mS,  1mS steps |
| | 1–16383Sec, 1Sec steps |
| | 1–16383Min, 1Min steps |

$x=60$μS for Delays#1–#2

$x=94$μS for Delay#3

$x=109$μS for Delays#4–8

$x=x+44$μS$-y$ for a recycle

delay.
if $n \geqslant 3$ and $m \leqslant 3$ then
$y=34\mu S$ otherwise $y=0$.

Minimum & maximum recycle count values:  1–65535, steps of 1 or unlimited.

Minimum & maximum external clock frequencies:  DC–4MHz guaranteed with current 4MHz rated PIC chips. Frequencies 10MHz–16MHz will probably work too. 10MHz rated PIC chips may be needed for frequencies >10MHz. Best to test external clock frequencies >4MHz before using on live experiment.

(4) Bugs

(a) The most complicated section of the software deals with scrolling of the LCD 'window' within the virtual editor display. It is difficult to eliminate all bugs from such code. Occasionally after certain edits the LCD is known to show erroneous entries such as something like;

```
P1=1μS
P1=1μS
P2=123μS
P3=1234μS
```

where the P1 pulse label is incorrectly written twice. This a benign bug and can be cleared by scrolling up/down the display causing the bad section to be repainted. If this is a problem, please make a note of the exact circumstances under which it occurs to aid debugging.

(b) For pulse sequences containing no *long*† pulse or delay values, pulse generation cannot be stopped with the <Start/Stop> key – Only a hard–reset will halt pulse generation. Note: Only one pulse or delay (eg. the last/repetition-time delay) in the pulse sequence needs to be *long* to avoid this bug,
† 'Long' means $> 32\mu S$ for pulses and typically $\geq 150\mu S$ for delays.

(c) Due to the fact that PICs internally divide their clock input frequency by a factor of 4, it is possible that there will be small differences in time between the start-of-pulse-generation-time on the master and any of the three slave boards. These differences will always be less than 1 machine cycle (ie. $<1\mu S$ with the internal 4MHz clock) and will remain constant until the pulse generator is Reset or switched off/on again – that is, it will not result in pulse jitter. More recent versions of the PIC pulse generator have extra H/W to

eliminate this problem. If this is a problem on older (eg. the 19" rack mounted units) please consult Alan Betts regarding possible modification.

(5) Possible Future Enhancements

- If RF noise from the system is a problem it may be possible to use the PIC's SLEEP instruction to put them into a dormant state when pulses are not being generated or the system accessed by a user etc.