

LP120
DEVELOPERS' GUIDE

2020.08.08

WRITING SOFTWARE DRIVERS
AND
DESIGNING PROGRAMMING-MODULES
FOR THE LP120

Lucid Technologies
<http://www.lucidtechnologies.info>
info@lucidtechnologies.info

Copyright (C) 1992-2007 by Lucid Technologies
All rights reserved.

The information in this manual has been carefully checked and is believed to be accurate. However, Lucid Technologies makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document. Lucid Technologies reserves the right to make changes in the products contained in this manual in order to improve design or performance and to supply the best possible product. Lucid Technologies assumes no liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights, nor the rights of others.

CONTENTS

1.0	General Information
1.1	Introduction
1.2	Hardware Design
1.3	Software Design
2.0	DRVR-XX.ASM
2.1	LP120 Version Compatibility
2.2	Program Flow
2.3	LP120 driver extension
3.0	Software Toolbox
3.1	Toolbox Equates
3.2	Description Syntax
3.3	Subroutine Definitions
3.4	Jump Definitions
4.0	Example Drivers
4.1	Hello world
Appendix A	DRVR-XX.ASM
Appendix B	HELLO.ASM

1.0 General Information

1.1 Introduction

This guide is an informal compilation of material for use in developing programs to run on the LP120 programmer. This is not a standalone document, several other documents contain information you will need. The LP120 User's Manual can be downloaded from the Data & Documentation page at www.lucidtechnologies.info. The manual includes the schematics for the LP120., you already have printed schematics and the LP120 USER'S MANUAL on disk. Along with this file you should also have received DRVR-XX.ASM (where XX is the current revision number) and LP120TST.ZIP.

1.2 Hardware Design

The LP120 was designed to use a Motorola 6803, 8-bit processor, operated in mode 2. The parallel ports are Motorola 6821 PIAs (Peripheral Interface Adapters). If you are unfamiliar with these parts there are many books available on the 6800 family of parts published by Motorola and others. The 6803 instruction set and 6821 datasheet can be downloaded from the Data & Documentation page at www.lucidtechnologies.info.

Parts being programmed must have a programming-module (PM) that plugs into the LP120's programming connector. A PM may be nothing more than a socket adapter or it can have complex circuitry of its own. The specifics of each PM depend on the electrical interface specifications of the part being programmed.

1.2.1 Power Control

The four power lines going to the programming connector are all controlled by the LP120. All four can be switched on and off under program control but only Vpp and Vps are programmable.

Port 1, bit 3, on the 6803 is called /SWVCC. /SWVCC switches 5V to the programming-module, PMVcc. PMVcc should be used to power 5V components on the programming-module other than the device being programmed. The maximum drain on PMVcc is 100 milliamps. When /SWVCC is low, PMVcc is on; when /SWVCC is high, PMVcc is floating.

Port 1, bit 4, on the 6803 is called /SWVFW. /SWVFW switches the unregulated full-wave rectified voltage to the programming-module, PMVfw. Since PMVfw is unregulated, it can vary from 15V to as low as 10V in response to the total load on the power supply. PMVfw can be used to supplement the other switched voltages on the PM. Limit the maximum drain on PMVfw to 500 milliamps. When /SWVFW is low, PMVfw is on; when /SWVFW is high, PMVfw is floating.

Schematic page 7 shows the MAX522, a write-only 8-bit dual output digital-to-analog converter (DAC). It is the reference for the DC-DC converter circuits that generate Vpp and Vps. Subroutines are provided in the toolbox for setting both these voltages.

Vpp should be used as the programming pulse voltage. Vpp can be set from 4.8 to 25.5 volts in 0.1 volt steps; settings below 4.8 volts are unreliable. A setting of 0 will effectively turn off Vpp. The current limitation of the Vpp supply is approximated by the equation given below.

$$I_{pp}(\text{amps}) \leq -0.262 + 0.299e^{\left(\frac{4.8}{V_{PP}}\right)} - 21.99e^{(-V_{PP})}$$

Vps should be used as the voltage powering the device in the programming socket, some devices require different supply voltages for programming and verification. Any other circuitry on the programming-module that must run on the same voltage can also be powered from Vps. Vps can be set from 1.8 to 7.2 volts in 0.03 volt steps; settings outside this range are unreliable. A setting of 0 will effectively turn off Vps. The current limits for the Vps supply are given below.

For $V_{ps} \leq 6.0V$, $I_{ps} \leq 0.5amp$

For $V_{ps} > 6.0V$, $I_{ps} \leq 0.5 - 0.072(V_{ps} - 6)amp$

Because Vpp and Vps are generated by switching circuits their rise-time is on the order of 1 to 4 milliseconds. In most cases the rise time of the voltage to the device being programmed is not important as the part can be held in reset, or inactive, until the voltage stabilizes. However, if the rise or fall time is critical, it may be necessary to use a transistor switch on the PM.

1.2.2 Programming-Modules

Programming-modules are based on standard 44 contact (0.156" spacing) plug-in prototype boards, such as:

- Radio Shack RSU10524486,
- Vector 3662-5,
- Lucid Technologies PM-WW.

Odd numbered contacts are on the component side of the board.

Be sure to consider the load you are putting on the ports of the 6821s when designing programming-modules.

All programming-modules should have a red LED that lights up whenever PMVcc is on. This LED indicates power is applied to the programming-module. The device being programmed should never be removed or inserted when this LED is on.

Whenever a programmable device is inserted or removed from the programming-module all signals going to it should be as close to ground potential as possible. PMVfw, PMVcc, Vpp, and Vps should all be turned off. Some ports on the PIAs have internal pullups so parallel ports should be changed to outputs and set low. Subroutines to accomplish this are provided as part of the software toolbox.

Most of you will be using DIP packages and you'll find the necessary ZIF (Zero-Insertion-Force) sockets are very expensive. If you plan on building several personality-modules you can sink a lot of money into ZIF sockets. It's more economical to buy one ZIF in each size you'll need and move it from one programming-module to another. The Aries (black) ZIF sockets (available in 24, 28, and 40 pins) will plug straight into a machine pin socket. The 3M/Textool (green) ZIF sockets have larger pins that won't fit a machine pin socket. However, you can solder a header onto the pins of the textool socket. This makes the socket ride higher, but it can now be moved from one machine pin socket to another.

1.3 Software Design

Although the LP120 was designed with the Motorola 6803 in mind, it will work with a 6801, 6803 or 6303R. A 6801 is a 6803 with internal ROM. In mode 2 a 6801 ignores its internal ROM

and functions exactly like a 6803. In mode 2, a Hitachi 6303R works like a 6803, with two important differences. First, the 6303R executes some instructions faster than the 6803. This means critical code may execute faster than expected on a 6303R. The software delay routines in the toolbox were written to minimize differences between the 6803 and 6303R. Second, the 6303R executes all 6803 op-codes plus a few unique to the 6303R. Therefore all LP120 code should be written and assembled for a 6803 target.

Any cross-assembler that generates code for the 6801 or 6803 should be acceptable. A 6801 freeware cross-assembler that runs under MS-DOS can be downloaded from the Data & Documentation page at www.lucidtechnologies.info. This software is freeware originally made available on the Motorola Freeware BBS.

1.3.1 Memory Utilization

The table below shows the LP120 memory map.

\$0000-\$001F	6803 REGISTERS
\$0020-\$007F	EXTERNAL RAM, 62256
\$0080-\$00FF	INTERNAL RAM, 6803
\$0100-\$7FFF	EXTERNAL RAM, 62256
\$8000-\$9FFF	EXTERNAL RAM, 6264
\$A000-\$A3FF	UNUSED
\$A400-\$A7FF	PIA ZERO
\$A800-\$ABFF	PIA ONE
\$AC00-\$BFFF	UNUSED
\$C000-\$FFFF	EXTERNAL EPROM, 27128

Notice that RAM is continuous from \$0020 to \$9FFF. The top 1K of RAM (\$9C00-\$9FFF) is reserved for stack and system variables, the rest (\$0020-\$9BFF) is available for your use. The normal convention for drivers is to use RAM from \$0020 to \$00FF as variable storage and 'ORG' the executable code at \$0100. If you need a large buffer area use the memory above the executable code.

1.3.2 Uploading Drivers

The driver you write will be uploaded to the LP120 using the upload option in the opening menu. Your driver is uploaded as a Motorola S-record file. As your driver is uploaded, each S-record is checked for accuracy then stored at the absolute address in the record. Any error will cause the upload to abort. At the end of a successful upload one of two things will happen:

- 1) If the driver's S9-record has an address of \$0000, control will return to the LP120 main menu. From the main menu you can start your driver by selecting the jump to \$0100 option, assuming your driver follows the normal conventions.
- 2) If the S9-record has any address other than zero, it will transfer control to that address.

So, assuming your driver's executable code begins at \$0100 and you want it to run as soon as it uploads, delete the S9-record in the assembler's output file (*.S19) and insert the one shown below.

```
S9030000FC *Delete this original S9-record, ($0000)
```

S9030100FB *Insert this S9-record in its place, (\$0100)

1.3.3 Initial Conditions

Because drivers load from the opening menu, they will always see the following conditions:

- * All interrupts are disabled.
- * The stack is assigned to LP120 reserved RAM.
There is no need to reassign the stack pointer.
- * The SCI (Serial Communications Interface) and port 2 of the 6803 have been properly initialized for serial communications.
- * All power (PMVfw, PMVcc, Vpp, Vps) going to the programming-module is off.
- * All PIA ports are outputs and set low.

1.3.4 Interrupts

The 6803 looks for its interrupt vectors at fixed addresses in high memory. Because these addresses are in the LP120's EPROM, your driver program will not be able to use any interrupts.

2.0 DRVR-XX.ASM

The file DRVR-XX.ASM is reproduced in Appendix A. It will give you a valuable start on your assembly language source code. Its comments contain a wealth of critical information. There are also equates for the 6803 registers and toolbox routines.

2.1 LP120 Version Compatibility

All the toolbox routines are entered by doing a JSR (Jump to SubRoutine). For example:

```
JSR  HEXASC
```

would temporarily transfer control to address \$FFD4 in the LP120's EPROM. The code at this address is another JSR to the actual hex-to-ascii subroutine in EPROM. All the toolbox addresses in EPROM are just vectors that reroute control to the correct location in EPROM. Because only vectors are tied to fixed locations, the LP120 EPROM can be updated easily. Thus, future revisions of the LP120 firmware may move the actual subroutines without changing the addresses of the vectors. In this way you are assured the code you write will be compatible with future revisions of the LP120.

2.2 Program Flow

Your driver functions as a transient program in the LP120's RAM. The entire driver (except for stack) must be contained in available RAM, \$0020-\$9BFF. When the driver is done, do a JMP (Jump) to RESET (\$C000) which returns control to the firmware in EPROM. Don't worry about restoring the state of the LP120, the jump to reset will reinitialize all pointers and hardware registers.

2.3 LP120 driver extension

Drivers for a particular part should use that part's number as part of the filename. The file

extension for all LP120 drivers should be D12. For example, if you were writing a driver to program a XY256 you would first copy DRVR-XX.ASM to XY256-01.ASM. Then edit XY256-01.ASM to write your driver program. Assemble XY256-01.ASM which will produce the output XY256-01.S19. Now edit the last line, the S9-record, of XY256-01.S19 to insert the correct starting address for the driver. Save the edited S19 file as XY256-01.D12, the first version of your XY256 driver.

3.0 Software Toolbox

3.1 Toolbox equates

DRVR-XX.ASM in Appendix 1 contains the toolbox equates. These equates define the fixed address of the Jump to SubRoutine for each subroutine in the toolbox.

3.2 Description Syntax

Each toolbox subroutine is explained in detail on the following pages. The definition syntax is shown below.

SUBROUTINE: [Name.]
 PRELOAD: [Any data that must be loaded into registers prior to calling will be explained here.]
 ACTIONS: [The subroutine's actions in the order performed.]
 REGISTERS: [The state of registers returned by the subroutine.
 ? = The register is changed.
 NC = The register is not changed.]
 ERROR FLAG: [If errors are possible and detected, the carry bit is used as an error flag.]
 NOTES: [Notes on how to use the subroutine.]

3.3 Subroutine Definitions

SUBROUTINE: ADRHEX
 PRELOAD: None
 ACTIONS: Return address pointer to the HEX-record data.
 REGISTERS: A=NC, B=NC, X=pointer to HBYTES
 ERROR FLAG: None
 NOTES: Data is stored in the following order and format:
 HBYTES RMB 1 *data bytes in record, HEX
 HADR RMB 2 *address of data, HEX
 HTYPE RMB 1 *record type, HEX
 HDATA RMB 64 *data bytes, checksum, HEX

SUBROUTINE: ADRMOT
 PRELOAD: None

LP120 Developers' Guide

ACTIONS:	Return address pointer to the S-record data.
REGISTERS:	A=NC, B=NC, X=pointer to STYPE
ERROR FLAG:	None
NOTES:	Data is stored in the following order and format: STYPE RMB 1 *record type, 1 or 9, ASCII SBYTES RMB 1 *remaining bytes, HEX SADR RMB 2 *address of data, HEX SDATA RMB 64 *data bytes, checksum, HEX
SUBROUTINE:	ASCHEX
PRELOAD:	A=ASCII character
ACTIONS:	Convert ASCII character in A to hex and return it in A.
REGISTERS:	A=Hex, B=NC, X=NC
ERROR FLAG:	C=1 if ASCII character is not 0-9 or A-F.
NOTES:	Lowercase ASCII (a-f) are not allowed.
SUBROUTINE:	BINBCD
PRELOAD:	D=unsigned binary value
ACTIONS:	Convert binary value in D to five BCD nibbles, return the BCD nibbles packed in A, B, and X.
REGISTERS:	X = BCD ten-thousands in LS-nibble, 0 elsewhere. A = BCD thousands in MS-nibble, BCD hundreds in LS-nibble. B = BCD tens in MS-nibble, BCD units in LS-nibble.
ERROR FLAG:	None
NOTES:	If you want to convert a single byte to BCD, clear A (the MSB of D) and load the value in B (the LSB of D).
SUBROUTINE:	DLY_A
PRELOAD:	X=delay value, \$0-\$FFFF
ACTIONS:	Delay approximately 50 microseconds times the value in X.
REGISTERS:	A=0, B=NC, X=0
ERROR FLAG:	None
NOTES:	This routine is not affected by the type of MPU chip used in the LP120. Actual delay = ((46 * X) + 24) * (1.085 usec), not including the JSR calling DLY_A. Use this routine for long delays.
SUBROUTINE:	DLY_B
PRELOAD:	D=delay value, \$0-\$FFF0

LP120 Developers' Guide

ACTIONS:	Delay slightly more than the number of clock cycles in D.
REGISTERS:	A=?, B=?, X=NC
ERROR FLAG:	None
NOTES:	This routine is affected by the type of MPU chip used in the LP120. 6803 delay = $(8 * \text{RND}(D/8) + 48) * (1.085 \text{ usec})$, 6303R delay = $(8 * \text{RND}((D+1)/8) + 46) * (1.085 \text{ usec})$, not including the JSR calling DLY_B. RND() is the round-up function, i.e. $\text{RND}(1.125) = 2$. Use this routine for high resolution delays.
SUBROUTINE:	DWNHEX
PRELOAD:	Hex image of record header, data, and checksum.
ACTIONS:	Send hex-record prefix, convert the record to ASCII and send it to the host.
REGISTERS:	A=?, B=?, X=?
ERROR FLAG:	None
NOTES:	The entire record, starting at HBYTES, must be stored in RAM in hexadecimal, not ASCII, format. See ADRHEX for details. Use this routine for normal data records (type 00). Each record is preceded by a carriage-return and line-feed.
SUBROUTINE:	DWNMOT
PRELOAD:	Hex image of record header, data, and checksum.
ACTIONS:	Send S1-record prefix, convert the record to ASCII and send it to the host.
REGISTERS:	A=?, B=?, X=?
ERROR FLAG:	None
NOTES:	The entire record, starting at SBYTES, must be stored in RAM in hexadecimal, not ASCII, format. See ADRMOT for details. Use this routine for normal data records, not end of file (S9) records. Each record is preceded by a carriage-return and line-feed.
SUBROUTINE:	GETVPP
PRELOAD:	A=initial VPP setting
ACTIONS:	The VPP setting is converted to the equivalent voltage value (see VPPSET for equation) and displayed, the user is prompted to increase or decrease the voltage to the desired value.
REGISTERS:	A=final VPP setting, B=?, X=?
ERROR FLAG:	None
NOTES:	The setting is restricted to the valid range for Vpp: $4.8\text{V} \leq V_{pp} \leq 25.5\text{V}$. Actual Vpp output is unchanged by this routine.

SUBROUTINE: GETVPS
PRELOAD: A=initial VPS setting
ACTIONS: The VPS setting is converted to the equivalent voltage value (see VPSSET) and displayed, the user is prompted to increase or decrease the voltage to the desired value.
REGISTERS: A=final VPS setting, B=?, X=?
ERROR FLAG: None
NOTES: The setting is restricted to the valid range for Vps:
1.8V <= Vps <= 7.2V.
Actual Vps output is unchanged by this routine.

SUBROUTINE: HEXASC
PRELOAD: A=byte to be converted to ASCII
ACTIONS: Convert byte in A to two character ASCII equivalent.
REGISTERS: A=High nibble ASCII, B=Low nibble ASCII, X=NC
ERROR FLAG: None
NOTES: Only 0-9 and uppercase ASCII (A-F) are returned.

SUBROUTINE: LC2UC
PRELOAD: A=ASCII character
ACTIONS: If the ASCII character byte in A is lowercase (a=\$61 to z=\$7A), convert it to uppercase (A=\$41 to Z=\$5A), otherwise return it unchanged.
REGISTERS: A=uppercase ASCII character, B=NC, X=NC
ERROR FLAG: None
NOTES: None

SUBROUTINE: MSGOUT
PRELOAD: X=starting address of ASCII character string
ACTIONS: Transmit ASCII character string via SCI. Special characters:
A tilde (~) is transmitted as a carriage-return line-feed sequence.
A zero (\$00) byte terminates the transmission.
REGISTERS: A=?, B=NC, X=?
ERROR FLAG: None
NOTES: Use this subroutine to send messages to the host.

SUBROUTINE: PIADAT
PRELOAD: None
ACTIONS: Select all PIA output data registers.

LP120 Developers' Guide

REGISTERS: A=?, B=NC, X=NC
ERROR FLAG: None
NOTES: None

SUBROUTINE: PIADDR
PRELOAD: None
ACTIONS: Select all PIA data direction registers.
REGISTERS: A=?, B=NC, X=NC
ERROR FLAG: None
NOTES: None

SUBROUTINE: PIAOFF
PRELOAD: None
ACTIONS: Make all PIA lines outputs,
set all PIA outputs to 0 (including CA2 and CB2),
return with PIA data registers selected.
REGISTERS: A=?, B=NC, X=NC
ERROR FLAG: None
NOTES: Called when done interfacing with programming-module. CA1 and CB1, pins
7 and 25 on the programming connector, are input only lines and can't be set
low.

SUBROUTINE: PWROFF
PRELOAD: None
ACTIONS: Sets Vpp and Vps to 0 volts, turns off PMVfw and PMVcc.
REGISTERS: A=?, B=?, X=0
ERROR FLAG: None
NOTES: Called when done interfacing with programming-module.
Voltages will decay exponentially.

SUBROUTINE: RX4HEX
PRELOAD: None
ACTIONS: Receive 4 ASCII-hex characters from the host,
convert these to a hexadecimal word and return it in D.
REGISTERS: A=MS-byte, B=LS byte, X=NC
ERROR FLAG: C=1 if a non-hex ASCII character is entered.
NOTES: This allows the user to enter an address or other 16-bit value.

SUBROUTINE: RX3HEX
PRELOAD: A=ASCII character for most significant hex nibble of 16-bit word.
ACTIONS: Receive 3 ASCII-hex characters from the host, convert these to a hexadecimal word with the preloaded value from A as the MS-nibble, return the word in D.
REGISTERS: A=MS-byte, B=LS byte, X=NC
ERROR FLAG: C=1 if a non-hex ASCII character is entered.
NOTES: This allows the first character of a 4 character value to be checked for special characters, like ESC or CR, before receiving the rest of the word.

SUBROUTINE: RX2HEX
PRELOAD: None
ACTIONS: Receive 2 ASCII-hex characters from the host, convert these to a hexadecimal byte and return it in B.
REGISTERS: A=?, B=byte, X=NC
ERROR FLAG: C=1 if a non-hex ASCII character is entered.
NOTES: This allows the user to enter any byte value.

SUBROUTINE: RX1HEX
PRELOAD: A=ASCII character for most significant hex nibble of byte.
ACTIONS: Receive 1 ASCII-hex characters from the host, convert this to a hexadecimal byte with the preloaded value in A as the MS-nibble, return the byte in B.
REGISTERS: A=?, B=byte, X=NC
ERROR FLAG: C=1 if a non-hex character is entered.
NOTES: This allows the first character of a 2 character value to be checked for special characters, like ESC or CR, before receiving the rest of the byte.

SUBROUTINE: RXECHO
PRELOAD: None
ACTIONS: Wait for SCI to receive a byte, send received value back to host, return received byte in A.
REGISTERS: A=received byte, B=NC, X=NC
ERROR FLAG: None
NOTES: None

SUBROUTINE: RX_UC

PRELOAD: None
ACTIONS: Wait for SCI to receive a byte. Convert to uppercase ASCII.
Return received uppercase byte in A.
REGISTERS: A=received/uppercase byte, B=NC, X=NC
ERROR FLAG: None
NOTES: None

SUBROUTINE: RX_UCE
PRELOAD: None
ACTIONS: Wait for SCI to receive a byte. Convert to uppercase ASCII.
Send uppercase value back to host, and return it in A.
REGISTERS: A=received/uppercase byte, B=NC, X=NC
ERROR FLAG: None
NOTES: Can be used to receive and echo menu entries.

SUBROUTINE: RXWAIT
PRELOAD: None
ACTIONS: Wait for data coming in via SCI to end. End is
defined as 1.0 seconds with no receive activity.
REGISTERS: A=?, B=NC, X=0
ERROR FLAG: None
NOTES: Use this if an error is detected in a long upload
that makes the rest of the upload unusable.

SUBROUTINE: SCIRX
PRELOAD: None
ACTIONS: Wait for SCI to receive a byte, return received byte in A.
REGISTERS: A=received byte, B=NC, X=NC
ERROR FLAG: None
NOTES: None

SUBROUTINE: SCITX
PRELOAD: A=byte to be transmitted
ACTIONS: Load byte into transmit register, wait for byte to be sent.
REGISTERS: A=NC, B=?, X=NC
ERROR FLAG: None
NOTES: None

SUBROUTINE: TX2ASC
PRELOAD: A=byte to be sent as ASCII-hex
ACTIONS: Translate the byte in A into two ASCII-hex characters,
send these ASCII characters via the SCI followed by an ASCII space.
REGISTERS: A=?, B=?, X=?
ERROR FLAG: None
NOTES: Use to display single byte data in hex format.

SUBROUTINE: TX4ASC
PRELOAD: D=word to be sent as ASCII-hex
ACTIONS: Translate the word in D into four ASCII-hex characters,
send these ASCII characters via the SCI followed by an ASCII space.
REGISTERS: A=?, B=?, X=?
ERROR FLAG: None
NOTES: Use to display addresses or other 16-bit data in hex format.

SUBROUTINE: TXBBIN
PRELOAD: A=byte to be sent as ASCII-binary
ACTIONS: Translate the byte in A into eight ASCII-binary characters (0 or 1),
send these ASCII characters via the SCI.
REGISTERS: A=?, B=?, X=?
ERROR FLAG: None
NOTES: Use to display single byte data in binary format.

SUBROUTINE: UPHEX
PRELOAD: None
ACTIONS: Upload an Intel HEX-record from the host,
confirm the checksum, and return.
REGISTERS: A=?, B=?, X=?
ERROR FLAG: C=1 if; sync character not found, invalid record
type, non-hex ASCII character, bad checksum.
NOTES: A HEX-record is one line of a *.HEX file.
If an error is found a specific message will be displayed on the host.
The data from the HEX-record is stored in LP120
reserved RAM, see ADRHEX for details.

SUBROUTINE: UPMOT
PRELOAD: None
ACTIONS: Upload a Motorola S-record from the host,
confirm the checksum, and return.

LP120 Developers' Guide

REGISTERS: A=?, B=?, X=?
ERROR FLAG: C=1 if; sync character not found, invalid record type, non-hex ASCII character, bad checksum.
NOTES: An S-record is one line of an *.S19 file.
If an error is found a specific message will be displayed on the host.
The data from the S-record is stored in LP120 reserved RAM, see ADRMOT for details.

SUBROUTINE: VPPSET
PRELOAD: A=setting for VPP
ACTIONS: Apply linearization correction to value in A, load DACA with corrected value in A.
REGISTERS: A=?, B=?, X=?
ERROR FLAG: None
NOTES: $V_{pp}=(A)/10$ volts. This is the default means to set Vpp using the correction table in EPROM.
If you want DACA set to the exact value passed in A use VPP_NC.

SUBROUTINE: VPP_NC
PRELOAD: A=setting for VPP
ACTIONS: Load DACA with the value passed in A.
REGISTERS: A=?, B=NC, X=NC
ERROR FLAG: None
NOTES: This routine bypasses the linearization correction used by VPPSET.
Vpp setting will be more accurate if VPPSET is used.

SUBROUTINE: VPSSET
PRELOAD: A=setting for VPS
ACTIONS: Apply linearization correction to value in A, load DACB with corrected value in A.
REGISTERS: A=?, B=?, X=?
ERROR FLAG: None
NOTES: $V_{ps}=3*(A)/100$ volts. This is the default means to set Vps using the correction table in EPROM.
If you want DACB set to the exact value passed in A use VPS_NC.

SUBROUTINE: VPS_NC
PRELOAD: A=setting for VPS
ACTIONS: Load DACB with the value passed in A.

LP120 Developers' Guide

REGISTERS: A=?, B=NC, X=NC
ERROR FLAG: None
NOTES: This routine bypasses the linearization correction used by VPSSET.
Vps setting will be more accurate if VPSSET is used.

3.4 Jump Definitions

JUMP: EXITMM
PRELOAD: None
ACTIONS: Branch to the LP120 main menu.
NOTES: This jump can be used as a debugging aid for uploaded drivers.
This jump leaves RAM intact so it can then be examined using the display system memory option from the LP120 main menu.

JUMP: RESET
PRELOAD: None
ACTIONS: Reinitializes all LP120 hardware, 6803 registers,
and writes \$FF to all nonreserved RAM.
NOTES: A jump to RESET is the normal method of terminating a driver and
returning control to the LP120 firmware.

4.0 Example Drivers

All drivers, example or otherwise, begin with the comments and equates from the current driver header file: DRVR-XX.ASM. Near the bottom of this file there is a section of RAM (\$0020 to \$00FF) reserved for variable storage. Finally, the executable code starts at \$0100.

The *.D12 files are the edited S19 files from the assembler. The last line in each file was changed to "S9030100FB" which will cause an automatic jump to \$0100 after the driver loads.

4.1 Hello world

Many introductory programming texts begin with a simple program to write the words "Hello world!" on the screen. So, it seemed like a good idea to show just how easily this can be done using the LP120 toolbox subroutines. Appendix B has the HELLO.ASM source code. The source file, HELLO.ASM, has been heavily commented to explain how it works.

Appendix A - DRVR-XX.ASM

```
* FILE FORM FOR LP120 DRIVER PROGRAMS
* COPYRIGHT 1990-2020 BY LUCID TECHNOLOGIES
* WRITTEN BY BRIAN BEARD, OWNER AND PRESIDENT
* ALL RIGHTS RESERVED
*
*REVISION HISTORY
*1.0 - 22SEP97, INITIAL LP120 RELEASE
*1.1 - 12JAN99, REWRITE FOR FIRMWARE CHANGES 2.2 & 2.3
*1.2 - 18DEC00, NEW TOOLBOX EQUATES FOR FIRMWARE 2.4
*      LP120 SOFTWARE CHANGES 2.5-2.7 DO NOT AFFECT DRIVERS
*
*6801/6803/6303R MPUs MAY ALL BE USED IN MODE 2.
*PORTS 3 AND 4 COMPRISE A MULTIPLEXED ADDRESS AND DATA BUS.
*CRYSTAL = 3.6864 MHZ, E = 921600 HZ, 1.085 uSEC PER CYCLE.
*EXECUTABLE PROGRAMS MUST LOAD TO RAM BETWEEN $0020 AND THE
* THE LP120 RESERVED RAM (RESRAM).  IF OTHER THAN $0100, THE
* START ADDRESS MUST BE CONTAINED IN THE S9 RECORD.  THIS MAY
* REQUIRE EDITING THE OUTPUT OF THE ASSEMBLER.
*
*MEMORY MAP (MODE 2) -----
* $0000-$001F   MPU REGISTERS
* $0020-$007F   EXTERNAL RAM, 62256
* $0080-$00FF   INTERNAL RAM, 6803/6303R
* $0100-$7FFF   EXTERNAL RAM, 62256
* $8000-$9FFF   EXTERNAL RAM, 6264
* $A000-$A3FF   UNUSED
* $A400-$A7FF   PIA ZERO
* $A800-$ABFF   PIA ONE
* $AC00-$BFFF   UNUSED
* $C000-$FFFF   EXTERNAL EPROM, 27128
*
*MPU PORT AND I/O USAGE -----
*PORT 1
* P10   MAX522 CHIP SELECT, ACTIVE LOW      OUTPUT  PULLUP
* P11   MAX522 SERIAL CLOCK                 OUTPUT  PULLUP
* P12   MAX522 SERIAL DATA                 OUTPUT  PULLUP
* P13   PMVcc SWITCH, ACTIVE LOW            OUTPUT  PULLUP
* P14   PMVfw SWITCH, ACTIVE LOW            OUTPUT  PULLUP
* P15   UNUSED                               PULLUP
* P16   UNUSED                               PULLUP
* P17   UNUSED                               PULLUP
```

LP120 Developers' Guide

```
*
*PORT 2
* P20   MODE SEL           INPUT   PULLUP
* P21   MODE SEL           INPUT   PULLUP
* P22   MODE SEL/EXTERNAL BAUD CLOCK INPUT
* P23   SCI RX DATA       INPUT
* P24   SCI TX DATA       OUTPUT
*
*INTERRUPTS
* NMI   UNUSED            INPUT   PULLUP
* IRQ   PIA-0             INPUT   PULLUP
*
*MPU (6803) REGISTER EQUATES -----
P1DDR   EQU    $00        *PORT 1 DATA DIRECTION REGISTER
P2DDR   EQU    $01        *PORT 2 DATA DIRECTION REGISTER
P1DAT   EQU    $02        *PORT 1 DATA REGISTER
P2DAT   EQU    $03        *PORT 2 DATA REGISTER
TCSR    EQU    $08        *TIMER CONTROL AND STATUS REGISTER
CNTR    EQU    $09        *2-BYTE COUNTER VALUE, $09=MSB
OUTCR1  EQU    $0B        *2-BYTE OUTPUT COMPARE REGISTER
INCAP   EQU    $0D        *2-BYTE INPUT CAPTURE REGISTER
RMCR    EQU    $10        *RATE AND MODE CONTROL REGISTER
TRCSR   EQU    $11        *TX/RX CONTROL AND STATUS REGISTER
RXDAT   EQU    $12        *RECEIVE DATA REGISTER
TXDAT   EQU    $13        *TRANSMIT DATA REGISTER
*
*PIA (6821) ADDRESS EQUATES -----
* A-SIDE PORTS HAVE INTERNAL PULLUP RESISTORS.
P0A     EQU    $A400      *PIA-0 DDR OR DATA FOR A-SIDE
P0ACR   EQU    $A401      *PIA-0 CONTROL REGISTER FOR A-SIDE
P0B     EQU    $A402
P0BCR   EQU    $A403
P1A     EQU    $A800
P1ACR   EQU    $A801
P1B     EQU    $A802
P1BCR   EQU    $A803
*
*MISC. EQUATES -----
TRUE    EQU    $FF
FALSE   EQU    $00
*
*ASCII EQUATES -----
BELL    EQU    $07        *BEEP
```

LP120 Developers' Guide

BS	EQU	\$08	*BACK SPACE
LF	EQU	\$0A	*LINE FEED
CR	EQU	\$0D	*CARRIAGE RETURN
ESC	EQU	\$1B	*ESCAPE
SPACE	EQU	\$20	*SPACE CHARACTER

*

*TOOLBOX EQUATES -----

RESET	EQU	\$C000	*JMP HERE TO TERMINATE DRIVER
RX_UCE	EQU	\$FF5C	*JSR
RX_UC	EQU	\$FF60	*JSR
LC2UC	EQU	\$FF64	*JSR
TXBBIN	EQU	\$FF68	*JSR
DLY_B	EQU	\$FF6C	*JSR
GETVPP	EQU	\$FF70	*JSR
GETVPS	EQU	\$FF74	*JSR
DWNMOT	EQU	\$FF78	*JSR
DWNHEX	EQU	\$FF7C	*JSR
RX4HEX	EQU	\$FF80	*JSR
RX3HEX	EQU	\$FF84	*JSR
RX2HEX	EQU	\$FF88	*JSR
RX1HEX	EQU	\$FF8C	*JSR
TX2ASC	EQU	\$FF90	*JSR
TX4ASC	EQU	\$FF94	*JSR
EXITMM	EQU	\$FF98	*JMP HERE FOR LP120 MAIN MENU
BINBCD	EQU	\$FF9C	*JSR
UPMOT	EQU	\$FFA0	*JSR
ADRMOT	EQU	\$FFA4	*JSR
UPHEX	EQU	\$FFA8	*JSR
ADRHEX	EQU	\$FFAC	*JSR
VPPSET	EQU	\$FFB0	*JSR
VPP_NC	EQU	\$FFB4	*JSR
VPSET	EQU	\$FFB8	*JSR
VPS_NC	EQU	\$FFBC	*JSR
PWROFF	EQU	\$FFC0	*JSR
PIAOFF	EQU	\$FFC4	*JSR
PIADAT	EQU	\$FFC8	*JSR
PIADDR	EQU	\$FFCC	*JSR
ASCHEX	EQU	\$FFD0	*JSR
HEXASC	EQU	\$FFD4	*JSR
MSGOUT	EQU	\$FFD8	*JSR
SCITX	EQU	\$FFDC	*JSR
SCIRX	EQU	\$FFE0	*JSR
RXECHO	EQU	\$FFE4	*JSR

LP120 Developers' Guide

```
RXWAIT EQU    $FFE8    *JSR
DLY_A   EQU    $FFEC    *JSR - APPROX (50 uSEC)*X
*
*RESTRICTED AREA -----
RESRAM  EQU    $9C00    *START OF LP120 RESERVED RAM,
      ORG    RESRAM * FOR SYSTEM VARIABLES, STACK, ETC.
      RMB    $03FF    * $9C00-$9FFF = 1K
*
*===== VARIABLE STORAGE =====
      ORG    $0020
*
*===== EXECUTABLE CODE =====
      ORG    $0100
```

Appendix B - HELLO.ASM

** Text from DRV.R.ASM would be here*

*===== EXECUTABLE CODE =====

ORG \$0100

*This line tells the assembler to start placing the code at
*address 0100 hex.

*The \$ tells the assembler this is a hex number.

*

LDX #MSG01 *POINTER TO MESSAGE_01

*This line loads the 16-bit index register (X) with the address
*MSG01 rather than the data at the label MSG01.

*The # tells the assembler to use the IMMEDIATE address mode.

*

JSR MSGOUT *SEND MESSAGE_01 TO THE HOST

*This Jump to SubRoutine line calls the toolbox routine MSGOUT.

*

JMP RESET

*This line causes control to jump back to the LP120 firmware.

*

*THE "HELLO WORLD!" MESSAGE

*The first line will generate the ASCII characters for the
*data between the single-quotes.

*The assembler directive FCC means (Form Constant Character).

*The tilde characters (~) cause MSGOUT to send a

*Carriage-Return/Line-Feed sequence at the beginning and end
*of the message.

*The second line generates a 00 byte which marks the end of
*the message string.

*The assembler directive FCB means (Form Constant Byte).

```
MSG01  FCC    '~Hello world!~'
        FCB    0
```