LRA, Vol. 4, pp. 183–192

Control of the Zymate Robot with an External Computer: Construction of a Multiple Peptide Synthesizer

Ronald N. Zuckermann,† Michael A. Siani,* and Steven C. Banville**

A multiple peptide synthesizer has been constructed by integrating a Zymate robot arm and a peptide synthesis station of our own design under the control of an Apple Macintosh II computer. The Macintosh software coordinates the movements of the Zymark robot arm, the switching of over 40 solenoid vales, and the monitoring of sensors. The robot arm is used to deliver solvent and reagents to an array of reaction vessels, and the Macintosh coordinates liquid flow via a series of solenoid valves. The Macintosh was chosen because it permits a friendly user-interface and supports powerful programming languages. THINK C language has been used to conveniently accommodate peptide sequence, position, and quantity information in complex data structures which are not supported by the EasyLab environment. A powerful communication protocol between the Macintosh and the Zymark controller is described.

INTRODUCTION

Peptides play an ever increasing role in biomedical and chemical research yet are quite costly to obtain. The synthesis of peptides has been automated but has been traditionally limited to the synthesis of one peptide at a time. Recently, however, automated multiple peptide synthesizers have been constructed that are capable of the simultaneous synthesis of up to 96 peptides [1, 2]. Such instruments not only reduce costs but dramatically increase the rate at which peptides can be generated.

A multiple peptide synthesis workstation has been constructed around a Zymate XP robot (Zymark Corp., Hopkinton, MA) that is capable of the simultaneous synthesis of 36 different peptides [3] as well as the synthesis of equimolar peptide mixtures [4]. We describe here a novel configuration for the control of the Zymate robot with an external computer. Specifically, a communication protocol between the external computer and the Zymark controller allows a significant enhancement of the robot's capabilities. This configuration is well suited for tasks such as multiple peptide synthesis that require controlling many external switches and performing calculations with multidimensional arrays. In addition, the configuration allows control of the workstation components independently from movement of the robot arm.

The synthesis of an array of peptides can be accomplished robotically by automation of the following tasks: repetitive pipeting of the appropriate amino acid solutions, dispensing of solvents, control of a large number of solenoid valves to control liquid flow, and the sensing of both digital and analog inputs. The ability to independently access an array of 36

*Ronald N. Zuckermann, Michael A. Siani, and Steven C. Banville, Chiron Corp., 4560 Horton St., Emeryville, CA 94608.
†Author to whom correspondence should be addressed.

Manuscript received 11/30/91; accepted 1/1/92

0895-7533/92/$3.50 + .25

different peptide synthesis reactions requires a robotic arm. We have chosen the Zymate XP robot, because its flexibility in changing hands allows all of these functions to be performed. In addition to the peptide synthesis applications mentioned, the Zymate system has also been configured to automatically cleave/deprotect the peptides after their synthesis, which is described elsewhere [5].

PEPTIDE SYNTHESIS WORKSTATION

The robotic workstation for multiple peptide synthesis consists of a 6 × 6 array of reaction vessels, a rack to store amino acid solutions, pressured solvent lines, and a syringe pump, all of which are accessible by the robot arm, as shown in Figure 1. The synthesis rack itself consists of fritted reaction vessels, Teflon™ tubing, a vacuum manifold, a pressurized argon manifold, and Teflon™ solenoid valves. The vacuum and argon manifolds are hooked up to each row of reaction vessels through the solenoid valves. By energizing the appropriate valve, liquid in the reaction vessels can either be mixed by opening to argon pressure (7–8 psi), causing gas to bubble through the fritted bottom, or removed by opening to vacuum (28–29 in. Hg), causing the liquid to drain. All of the liquid that is drained out of the reaction vessels is collected in a 4 L vacuum trap. Periodically, the solvent in the vacuum trap is automatically delivered to a larger 25 L container with argon pressure, avoiding the need to evacuate a large vacuum trap.

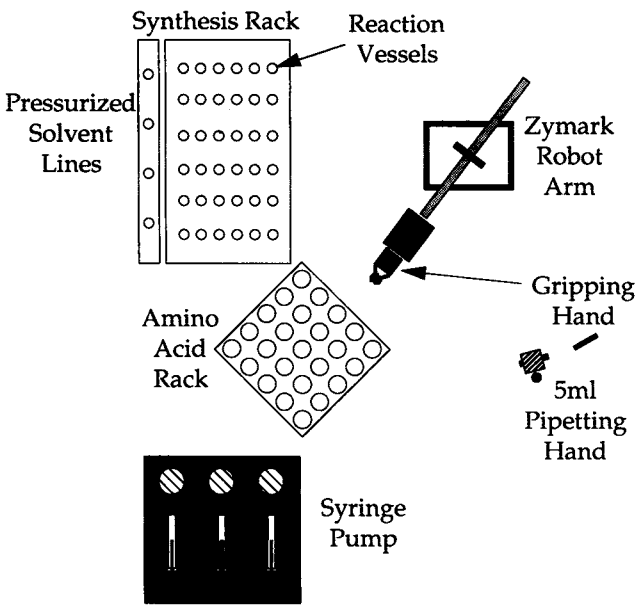

Figure 1. The robotic work station consists of a 6 × 6 array of reaction vessels, a rack to store amino acid solutions, pressurized solvent lines, a robotic arm, a 5 mL syringe hand, a gripping hand, and a syringe pump.

The solenoid valves that control these functions are operated under the control of a Macintosh II computer (Apple Computer, Cupertino, CA) equipped with an internal multifunction analog and digital I/O board (MacADIOS II board, GW Instruments, Inc., Somerville, MA). Attached to this board are three daughter boards that increase the number of inputs and outputs to more than 48. Since each Power and Event Controller provided for this purpose by Zymark Corp. can only control eight outputs, an application which requires a large number of digital inputs and outputs can be more easily and economically constructed by including an external computer. Ribbon cables from the multifunction I/O board are plugged into a circuit board (Newark, San Francisco, CA) containing an array of solid state relays (Newark). The multifunction I/O board triggers the relays which connect the solenoid valves to a +24 V dc power supply by outputting 0V and +5V signals.

Analog inputs on the MacADIOS II board are used for sensing reagent delivery pressure, system vacuum, and level of liquid in reagent containers. The vacuum sensor and the pressure sensor (both from Cole-Parmer, Chicago, IL) are transducers that have an applied voltage of +13.8 V dc and an output ranging from +1–6 V dc. The output voltage is read by the analog input portion of the MacADIOS II board and is converted by the software to a pressure or vacuum reading. The liquid level sensors were constructed from reflective object sensors which consist of an infrared emitter and sensor pair (Instruments for Research and Industry, Cheltenham, PA). This type of sensor can detect the liquid level from the outside of a glass bottle, eliminating any contact with solvents. The amount of light reflected from the bottle is affected by the presence or absence of liquid in the bottle. When any of the solvents fall below a specified level, the computer will pause the synthesis until the bottle is refilled.

Delivery of solvents and reagents to the reaction vessels is accomplished by a combination of robot arm movements and solenoid valve control. Wash solvents are distributed to the reaction vessels with a solvent line from a pressurized (7–8 psi) bottle under control of a solenoid valve. The solvent line or spigot is moved to the appropriate reaction vessel with the general purpose gripping hand, and the solenoid valve is momentarily energized.

The amino acid and capping solutions are stored in a 5 × 5 array of 250 mL polypropylene conical centrifuge tubes. Each reagent solution is transferred

to the appropriate reaction vessel with a 5 mL pipet hand. A separate 5 mL pipet tip is used for each reagent to avoid cross contamination. Small amounts of activating reagent (50–100 μL) are accurately dispensed with a spigot line connected to a multichannel syringe pump (Master Laboratory Station, Zymark Corp.).

The order of operations required for the solid-phase synthesis of peptides with 9-fluorenylmethoxycarbonyl (Fmoc) chemistry [6] is shown schematically in Figure 2. The addition of each amino acid monomer to a growing polypeptide chain requires three steps: deprotection, coupling, and capping. First, the N-terminal Fmoc protecting group on the peptide-resin must be removed with a basic (piperidine) solution to expose a reactive amino group. The appropriate amino acid solution and activating reagents are then added to couple the new amino acid to the exposed amino group. Finally, any unreacted amino groups are capped (acetylated) to prevent chain elongation of these failed sequences.

The entire robotic synthesis area is under a 24-hour video surveillance. A video recorder (VCR) is interfaced with the MacADIOS II board through the

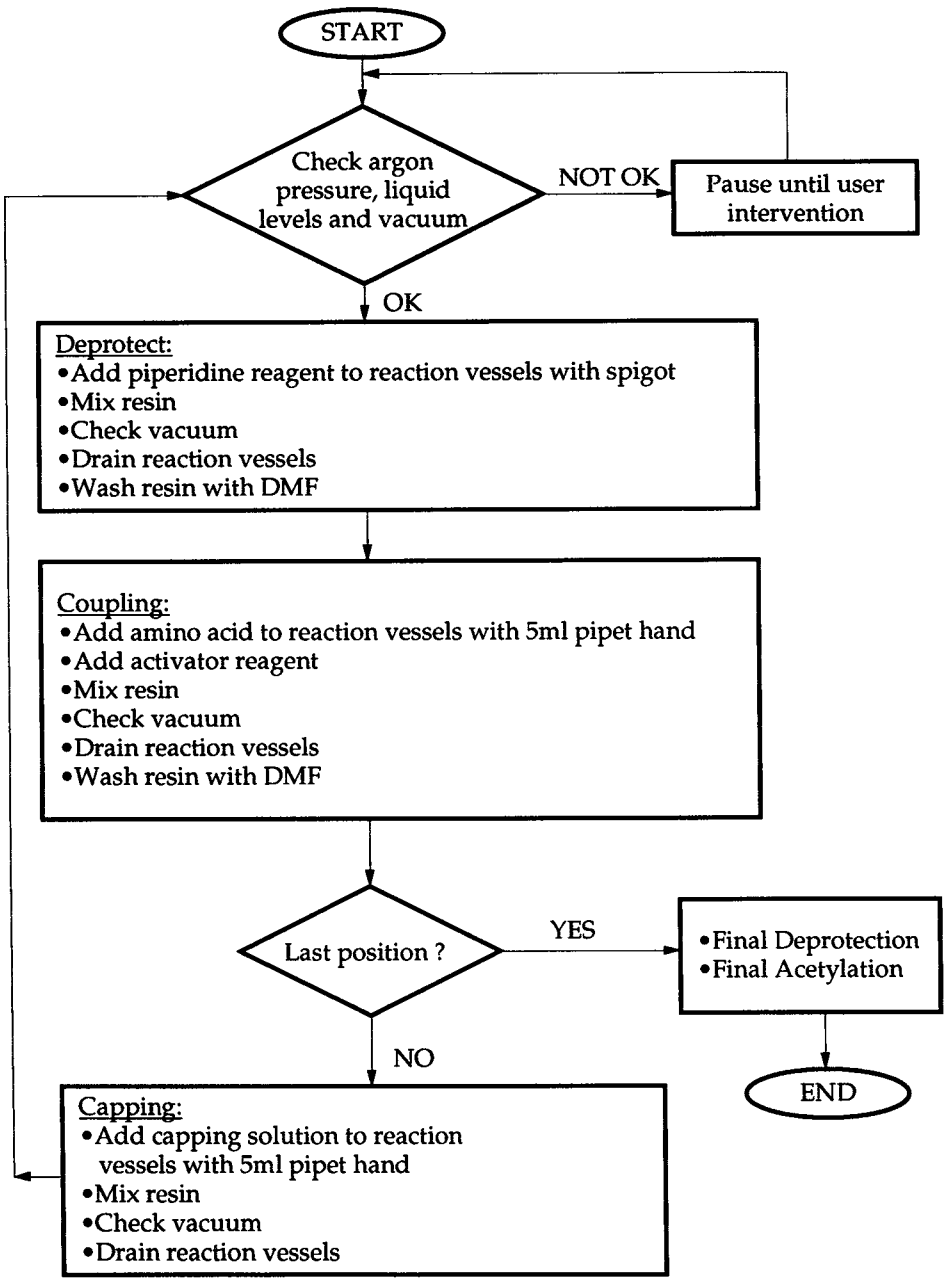

Figure 2. The addition of each amino acid monomer to a growing polypeptide chain requires three steps: deprotection, coupling, and capping.

VCR remote control. The Macintosh instructs the VCR to record only when there is movement of the robot arm. This allows ~18 hours of system operation to be recorded on an 8-hour tape. Should a problem occur while no one is present, review of the videotape helps troubleshoot the problem.

SOFTWARE

General Description

External control of the robotic multiple peptide synthesizer requires that a computer have the ability to read and write to data files, read input from the user, communicate with other computers, run a multifunction I/O board, and support a graphical, mouse-driven interface. A computer language that supports independent subroutines, complex data structures such as multi-dimensional arrays, local variables, and compiled code was also required. A Macintosh II computer and THINK C version 5.0 (Symantec Corp., Cupertino, CA) programming language were chosen to meet the above criteria. Some of these software tasks are possible with the Easylab language used by the Zymark controller but are cumbersome because multidimensional arrays are not supported and all variables are global.

Sequence information is read from external data files created by the user before the program is started. The complete set of input data includes the following: the list of peptide sequences, the reaction scale, the resin substitution, the molecular weights of the amino acids, and whether or not there should be a final deprotection and acetylation. After the data are read, the program assigns each reaction vessel an array of amino acid sequences and calculates the amount of amino acids and other reagents that are needed. The results of the calculations are then printed out and written to a log file.

As each major step of the synthesis is performed, the time, event description, and sensor status are written to a log file. The log file is written to a printer and the Macintosh hard disk. By writing the logfile to a second external computer, the robot's progress can be monitored off-site by calling in with a modem.

The MacADIOS II board is accessed from the THINK C program through libraries of high-level subroutines that are supplied by GW Instruments. The two subroutines used (cread and cwrite) allow data to be read and written to the I/O board.

A graphical, mouse-driven user interface was created within the Macintosh environment. Pull down menus and windows with buttons and text fields have been implemented to make data entry and manual operation of synthesizer functions more intuitive. For example, valves can be switched on and off in a manual mode by just selecting a button with the mouse. Pull down menus can be used to pause a program or change the course of the program without having to abort it. Menu items can also be created that will interact with the Zymark controller.

Macintosh-Zymark Communication Protocol

The interaction between the hardware and software is controlled through a Macintosh II computer. The Macintosh software is at the center of control (Figure 3), instructing the Zymark robot what to do at the appropriate time, energizing solenoid valves, monitoring input data from sensors, and performing calculations. The communication protocol between the Macintosh and the Zymark involves the two-way transmission of data, allowing the verification of each command sent to the robot. The THINK C code that runs on the Macintosh and controls the communication protocol is shown in the Appendix.

Communication is governed by a set of subroutines that send and process a stream of ASCII characters through the modem serial port of the Macintosh. The modem serial port is linked to a Remote Computer Interface Module (Zymark Corp., Hopkinton, MA) in the Zymark System V controller. These routines access the serial port through the Macintosh Communications Toolbox using calls directly from THINK C. At the start of the main program, the port is initialized by calling the *zymark_serialinit()* subroutine, which opens the port, sets the communication parameters (baud rate, number of data and stop bits, and parity), allocates a serial management buffer, and allocates space for the zymark input buffer. The baud rate is set to 2400 baud and is presently limited by the Zymark interface. The *zymark_flush_out_line()* subroutine is then called to clear the communications channel of any extraneous noise. Commands to the Zymark are then sent with the *zymark_send_command()* subroutine.

The *zymark_send_command()* subroutine breaks up the command into a stream of characters and sends it via the *zymark_write_word()* subroutine. The robot then executes the command (if it is recognized) and sends a reply back. The *zymark_send_command()* subroutine processes whatever is returned or echoed by the Zymark, allowing verification that the transmission was successfully received. Upon receipt of an "OK," *zymark_send_command()* returns to the main program. In the event

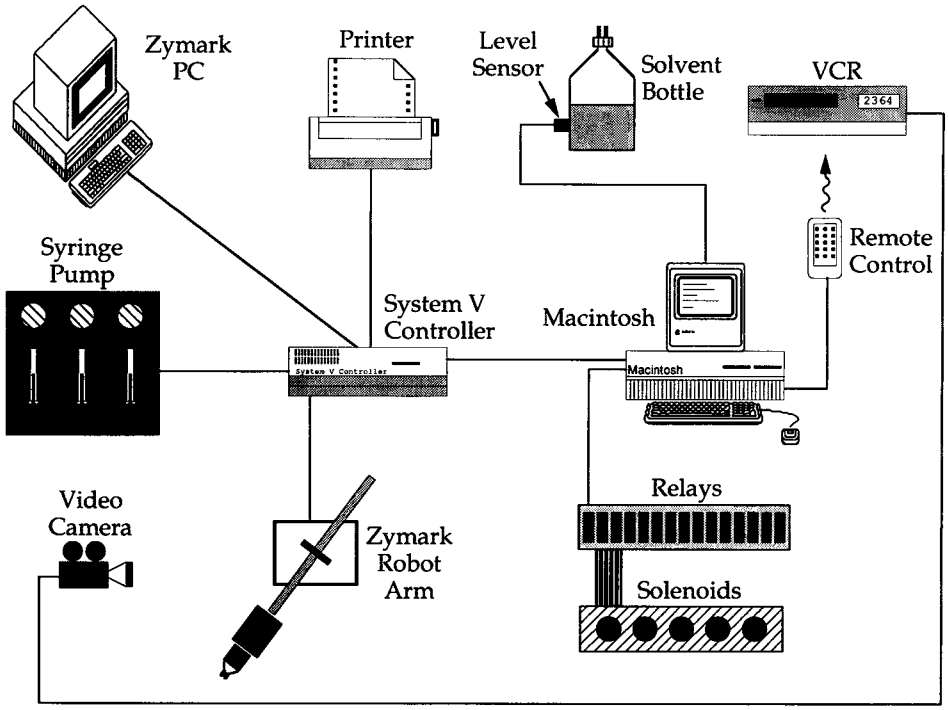

Figure 3. The Macintosh software is at the center of control, instructing the Zymark robot what to do at the appropriate time, energizing solenoid valves, monitoring input data from sensors, and performing calculations.

of a "NOTOK," the routine attempts to resend the command (up to three times) and if unsuccessful pauses until the user intervenes. However, we have found this to be a sufficient number of tries in all runs to date.

In some cases, we desire explicit information from the Zymark. For instance, when we want to be sure that the Zymark robot has grasped the spigot in its general purpose hand, we inquire of the Zymark the value of HAVE.SPIGOT.FLAG using the subroutine *zymark_request_integer()*. This subroutine simply puts the Zymark in a verbose mode by sending the command VERBOSE3, then concatenates the variable name to a question mark (e.g., "? HAVE.SPIGOT.FLAG"), then sends the query to the Zymark. The Zymark returns the value of the variable; in this case, "TRUE" if the spigot has been grasped successfully.

INTERACTION OF HARDWARE AND SOFTWARE

Examples of how this communication protocol is used during peptide synthesis are shown for a coupling step (Figure 4) and a washing step (Figure 5). The washing step, for example, begins with the Macintosh software instructing the Zymark robot arm to pick up the general purpose gripping hand, followed by a command to pick up the wash spigot. The Macintosh then calculates the index number of the first reaction vessel to receive wash solvent and instructs the robot arm to move to that position. Prior to liquid dispensing, the pressure sensor and liquid level sensors are read by the MacADIOS II board to confirm that liquid will in fact dispense at the desired flow rate. The Macintosh software then sends a command to the multifunction I/O board to energize the solenoid valve controlling the wash solvent for a specific period of time. In this way, a specified volume of wash solvent is delivered to a particular reaction vessel. The index number is then incremented and the new value sent to the robot instructing it to move to the new position. This process is repeated until all vessels in use have received liquid.

Mixing of the added solvent with the resin beads in each reaction vessel is necessary to assure thorough washing. The Macintosh software accomplishes this by periodically energizing the solenoid valve that connects each row of reaction vessels to pressure in the argon manifold. Mixing occurs from agitation caused by the gas bubbles, and therefore movement of the reaction vessels is not necessary. Prior to liquid removal, the system vacuum sensor is read by the MacADIOS II board to ensure proper drain rates. The solvent is then drained from the vessels when the Macintosh instructs the MacADIOS II board to energize the solenoid valves connecting them to the vacuum manifold. Finally, the Macintosh instructs the robot to return the spigot line to its storage rack and park the hand.

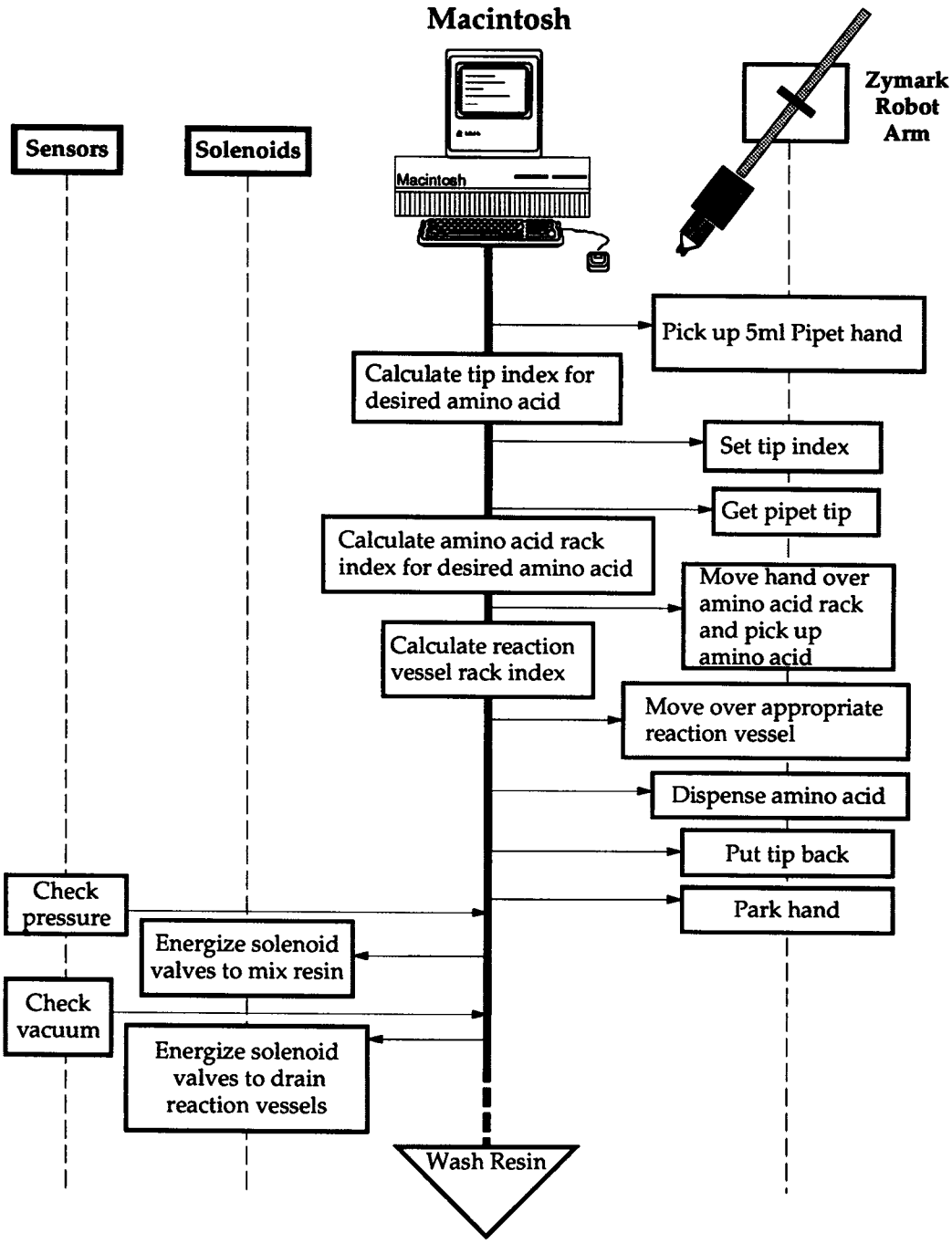

Figure 4. The Macintosh must communicate with the Zymark robot, solenoid valves, and sensors to perform the coupling step.

CONCLUSION

A multiple peptide synthesizer has been constructed by interfacing an external personal computer and a Zymate Laboratory Automation System. A Macintosh II, modified with a multifunction I/O board, serves as the center of control and coordinates the actions of the Zymark robot, relays, and solenoid valves. This configuration was chosen because it allows the convenient control of a large number of external devices and supports powerful programming languages such as THINK C. The communication protocol between the Macintosh and the Zymark controller presented here can facilitate sophisticated robotic applications that may not be possible with off-the-shelf systems.

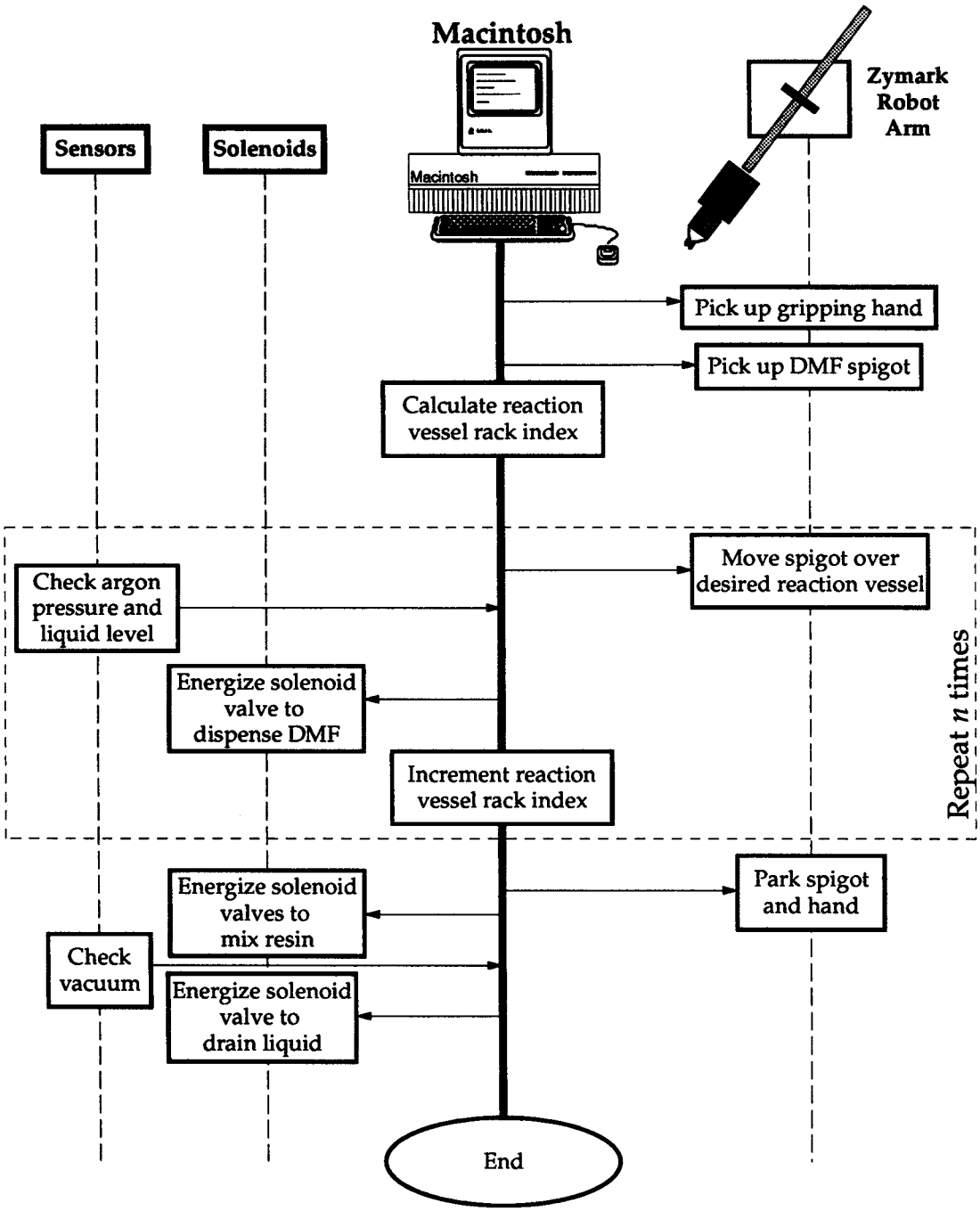

Figure 5. The Macintosh must communicate with the Zymark robot, solenoid valves, and sensors to perform the wash step. Certain tasks must be performed *n* times, where *n* represents the number of reaction vessels.

ACKNOWLEDGMENTS

The authors would like to thank Janice Kerr for valuable assistance, Alex Wong for assistance in software development, and Brian Warner for helpful discussions.

APPENDIX

Below is the THINK C code for the Macintosh-Zymark communications subroutines. The typedef ZYMARK_COMM contains fields for holding transmitted and received commands.

Appendix: page 1

```
/*              ---------------------------------------------------------------
                zymark_serialinit: Initialize serial, printer port for communications
                --------------------------------------------------------------- */
OSErr zymark_serialinit()
{
OSErr           err;
SerShk          flags;

                /* open serial port B */
        if(err = OpenDriver(OUTDRIVER, &goutRefNum)) return err;
        if(err = OpenDriver(INDRIVER, &ginRefNum)) return err;
                /* set communication parameters */
        if(err=SerReset (goutRefNum, baud2400+data8+stop10+noParity)) return err;
        if(err=SerReset (ginRefNum, baud2400+data8+stop10+noParity)) return err;
                /* give serial manager a bufsiz buffer */
        if(!(gzymark_comm.sermgrbuf = NewPtr(SERBUFSIZ))) return MemError();
        if(err = SerSetBuf(ginRefNum, gzymark_comm.sermgrbuf, SERBUFSIZ)) return err;
                /* handshaking stuff */
        flags.fXOn = TRUE;
        flags.fInX = TRUE;
        flags.xOn = XONCHAR;
        flags.xOff = XOFFCHAR;
        if(err=SerHShake(ginRefNum, &flags)) return err;
                /* allocate space for zymark input buffer */
        if(!(gzymark_comm.inbuf=(char *)NewPtr(SERBUFSIZ))) return MemError();
        return noErr;
}/*zymark_serialinit*/

/*              ---------------------------------------------------------------
                zymark_serialwrite: send a character through serial port to zymark.
                --------------------------------------------------------------- */
void  zymark_serialwrite(char ch)
{
long            num=1;

        (void) FSWrite(goutRefNum, &num, &ch);
}/*zymark_serialwrite*/

/*              ---------------------------------------------------------------
                zymark_getserialchars: get the stream of characters from the serial port.
                --------------------------------------------------------------- */
void  zymark_getserialchars(long  count)
{
        (void) FSRead(ginRefNum, &count,gzymark_comm.inbuf);
}/*zymark_getserialchars*/


/*              ---------------------------------------------------------------
                zymark_write_word: write a command to the zymark through the serial port.
                --------------------------------------------------------------- */
zymark_write_word(char *ccommand)
{
int             i=0;
char            ch;
                /* loop through characters in command; send through port */
        i=0; while(ccommand[i] != '\0') {
                ch=ccommand[i];
                zymark_serialwrite(ch);
                i++;
        }
                /* terminate command with carriage return/line feed. */
        ch=13; zymark_serialwrite(ch);
        ch=10; zymark_serialwrite(ch);
}/*zymark_write_word*/
```

Appendix: page 2

```
/*      ------------------------------------------------------------------
        zymark_send_command: Send zymark command and process the stream that comes
        back.  Depending on the level of verbosity, we can expect commands echoed and
        OK or NOTOK.
        ------------------------------------------------------------------ */
zymark_send_command(char *ccommand)
{
int             i=0,ii=0,retrycount=0,sendflag=TRUE;
long            count=0,starttick=0,currtick=0,maxtime=0;

        while(sendflag==TRUE){
                        /* send command through serial port to zymark */
                zymark_write_word(ccommand);
                        /* collect response from zymark until we overflow buffer */
                i=0;
                starttick = TickCount();
                while(i<smallbufsiz){
                        if((count = zymark_serialcharsavail()) > 0) {
                                /* some characters are available from zymark, get them */
                                zymark_getserialchars(count);

                                /* check for line-feed, carriage return */
                                if((gzymark_comm.tstr[i]==10) && (gzymark_comm.tstr[i-1]==13)){
                                        if(ends_with(gzymark_comm.tstr,"OK") == TRUE) {
                                                break;          /* got and OK or NOTOK */
                                        } else {
                                                                /* just clear buffer and get more */
                                                clean_buffer();
                                        }
                                } else { i++; }
                        }
                }
                        /* got a return, line feed, or too many characters */
                if((ends_with(gzymark_comm.tstr,"OK") == TRUE) && (i < smallbufsiz)) {
                        /* got an OK or a NOTOK */
                        if(ends_with(gzymark_comm.tstr,"NOTOK")==TRUE) {
                                /* got a NOTOK, wait 10 sec. before retrying (up to 3 times). */
                                starttick = TickCount();
                                if (retrycount < 3) {
                                        printf("i=%d, NOTOK, will retry in 10 seconds\n",i);
                                        retrycount++;
                                        sendflag=TRUE;
                                } else { sendflag=FALSE; }
                                while((TickCount() - starttick) < (60*10)) { }
                        } else {
                                /* got an OK */
                                sendflag=FALSE; return TRUE;
                        }
                } else {
                                /* buffer overflow. */
                        printf("%d: PROBLEM %s\n",retrycount,ccommand);
                                /* wait 10 seconds and resend up to 3 times */
                        starttick = TickCount();
                        while((TickCount() - starttick) < (60*10)) { }
                        if (retrycount < 3) { retrycount++; sendflag=TRUE;
                        } else { sendflag=FALSE; }
                }
        }
} /* zymark_send_command */
```

REFERENCES

[1] G. Schnorrenberg and H. Gerhardt: "Fully Automatic Simultaneous Multiple Peptide Synthesis in Micromolar Scale—Rapid Synthesis of a Series of Peptides for Screening in Biological Assays," Tetrahedron *45,* 7759–64 (1989).

[2] H. Gausepohl, M. Kraft, C. Boulin, and R. Frank, "Automated Multiple Peptide Synthesis with BOP Activation," in J. Rivier and G. Marshall, (ed) Peptides: Chemistry, Structure and Biology (Proceedings of the 11th American Peptide Symposium), ESCOM, Leiden, pp. 1003–04 (1990).

[3] R. Zuckermann, J. Kerr, M. Siani, and S. Banville: "Design, Construction and Application of a Fully Automated Equimolar Peptide Mixture Synthesizer," Int. J. Pept. Protein Res., in press (1992).

[4] R. Zuckermann, J. Kerr, M. Siani, S. Banville, and D. Santi: "Identification of Highest-Affinity Ligands by Affinity Selection from Equimolar Peptide Mixtures Generated by Robotic Synthesis," Proc. Natl. Acad. Sci. U.S.A. *89,* 4505–4509 (1992).

[5] R. Zuckermann and S. Banville: "Automated Peptide-Resin Deprotection/Cleavage by a Robotic Workstation," Peptide Res. *5,* 169–174 (1992).

[6] G. Fields and R. Noble: "Solid Phase Peptide Synthesis Utilizing 9-Fluorenylmethoxycarbonyl Amino Acids," Int. J. Pept. Protein Res. *35,* 161–214 (1990).