VLSI IMPLEMENTATION OF A NUMERICAL PROCESSOR FOR ROBOTICS*

J. L. Turney and T. N. Mudge

Department of Electrical and Computer Engineering
University of Michigan
Ann Arbor, MI 48109

ABSTRACT

This paper presents the preliminary specification for a very large scale integrated (VLSI) circuit implementation of a single chip processor for dedicated numerically intensive applications. In particular, the numerical processor is intended for the real-time control of a robot arm. The architecture of the processor is outlined, and preliminary timing figures are given. The control strategy for the control of a robot arm is discussed. Finally, the results of a functional simulation of the processor executing part of a control program are presented.

Keywords: Robotics, VLSI, Controller.

INTRODUCTION

This paper presents the preliminary specification for a very large scale integrated (VLSI) circuit implementation of a single chip processor for dedicated numerically intensive applications. Circuit densities commensurate with levels of integration projected for the mid-1980s are assumed. The proposed numerical processor (NP) is suitable for real-time control where sophisticated control strategies require very large numbers of high precision arithmetical operations to be performed for every input/output transaction. In particular, the NP is intended for the real-time control of a robot arm. The NP functions as an attached processor of a general purpose minicomputer. Conceptually, it lies between Floating Point Systems' AP120B [F179], a high performance numerically oriented attached processor, and the Intel 8087 [Pa80], a single chip numerically oriented attached processor in the Intel 8086 family of components [In79]. All three work with floating-point numbers. The NP differs from the AP120B by being much simpler, less flexible, slower, and by having a smaller word size (32 bits versus 38 bits). It differs from the 8086 by having its own on chip program memory, input/output buffers to facilitate real-time applications, and two independent function units. However, the 8087 has a more flexible number format,and can deal with several variants of the IEEE floating point standard up to and including the 80 bit format.

The motivation for this work arose during the planning of a multirobot assembly system at the University of Michigan [LM80]. This system is shown in Figure 1.

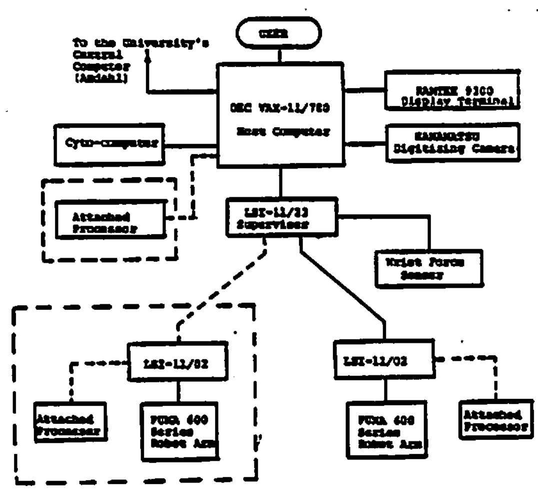

Figure 1. Multirobot System.

Most of the major hardware components of

*This work was supported in part by an unrestricted grant from Fairchild Camera and Instrument Corp.

Preprint from the 1981 International Instrumentation Symposium

this system are in place and operational. Those components shown surrounded by broken lines represent future extensions. Consideration of the computing needs of such a system leads to the idea of "application-directed" machines to perform the tasks of vision and real-time arm control. Processors to do this are shown in Figure 1 as blocks labeled "Attached Processor". The one at the top left is for vision and the other two are for robot arm control. The NP discussed in this paper is intended to satisfy the requirements of an attached processor for robot arm control. The additional specification that the NP be realized as a single chip processor arose out of a desire to develop a substantial project to test out the VLSI systems design program at the University of Michigan. However, a prototype will be constructed from low power Schottky TTL components to allow the robotics work to be decoupled from the VLSI work. This preliminary study assumes the NP will be implemented in nMOS because our present expertise is in this technology. However, our eventual aim is to investigate the design of the NP in a faster technology that still has the density of integration associated with nMOS. A prime candidate is the I3L (Isoplanar Integrated Injection Logic) technology developed by Fairchild Corporation.

This paper is organized as follows. The next section discusses the architecture of the NP. Section 3 discusses the intended application of the NP--robot arm control. Section 4 presents the simulation result from a functional simulation of the NP. Section 5 is the conclusion.

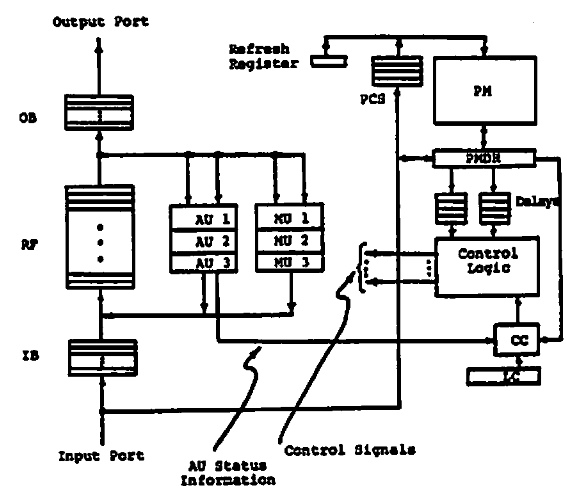

Figure 2. Numerical Processor.

ARCHITECTURE

Figure 2 shows a block diagram of the proposed NP. The major components are as follows:

1. A 32 bit floating point adder unit (AU).
2. A 32 bit floating point multiplier unit (MU).
3. A 256x32 register file (RF).
4. A 32x32 bit input buffer (IB).
5. A 32x32 bit output buffer (OB).
6. A 1Kx50 bit program memory (PM).
7. A 4x10 bit program counter stack (PCS).
8. A 1x50 bit program memory data register (PMDR).
9. A 16 bit loop counter (LC).
10. Condition code logic (CC).

A preliminary gate level logic design and layout of an nMOS realization of the chip, using the design rules given in [MC80], indicates that 50% of the area will be occupied by the AU, MU, RF and PM. The other components occupy less than 10% of the area, and the buses, control signal lines, and bonding pads occupy the remaining 40% of the chip. An estimate, based on a logic gate count, of the number of active devices required by the chip indicates that 90% will be contained in just four of the components--the AU, MU, RF, and PM. The estimate shows 16K devices are required for the AU, 32K for the MU, 16K for the RF, and 55K for the PM. The estimate for the total device count works out to be 150K. This is well within projections for single chip systems in the mid 1980's. At that time 1 million devices/chip are anticipated [PS80].

The floating point number format used in the design study is the 32 bit proposed IEEE standard described in [Co79]. In this format a normalized nonzero number X has the form:

$$X = (-1)^{S} \times 2^{E-127} \times 1.F$$

where,
- S = sign bit
- E = 8 bit exponent biased by 127
- F = 23 bit fraction which, together with an implicit leading 1, yields the

significant digit field "1.---".

Both the AU and the MU were designed to handle this normalized format. However, the rounding modes, rounding precision control, infinity arithmetic, denormalized arithmetic, most of the floating point exceptions, and the various extended formats called for by the proposed standard were not considered in the design of either the AU or the MU. Naturally, inclusion of any of these features would increase the complexity of both the AU and MU, and estimates of the device count would have to be adjusted accordingly.

The AU is a three stage pipeline with the first stage performing fraction alignment, the second stage performing fraction addition, and the final stage performing normalization. Alignment is performed using an 8 bit subtractor with full lookahead to determine the number of shifts needed followed by a 5 (= $\lceil \log_2 24 \rceil$) level 24 bit barrel shifter to execute the shifts. Fraction addition is performed using a standard 24 bit binary adder structure with partial carry lookahead across groupings of 4 bits. Normalization is performed using another 24 bit barrel shifter. The basic machine cycle (M-cycle) is targeted at 500 nS. Each stage of the pipeline completes its task within an M-cycle, thus when the AU is in streaming mode--operands are being fed to it as fast as possible--it produces a result every 500 nS. The AU is constructed from standard NOR/NOR PLAs (program logic arrays--see [MC80]) having an estimated delay of 50 nS. This is fast enough to be used as a building block in the construction of an alignment stage--potentially the most time consuming of the AU's three stages--that can operate within 500 nS. It is also fast enough to construct the binary adder for the fraction addition, as well as the barrel shifter for the normalization stage.

The MU is also a three stage pipeline with the first stage performing partial product generation and carry-save addition, the second stage performing carry propagation addition to produce the unnormalized 48 bit product fraction, and the final stage performing normalization, truncation to 24 bits, and exponent addition. The design of the multiplier is quite standard (see [Ku78]). Stage one uses a tree of 3-input to 2-output carry-save adders, implemented with PLAs as the basic building block. With a tree height of 8 (= $\lceil \log_{3/2} 24 \rceil$) and 50 nS delay per PLA the 500 nS time limit for a pipeline stage is easily met (generating the partial products requires only an array of AND's and adds only 15 nS to the delay time). Stage two uses a 48 bit adder with full lookahead. The lookahead is across groups of 4 bits, and is performed by lookahead units that are realized as PLAs. The lookahead units themselves produce group propagate and group generate signals which feed another level of lookahead units. This process is continued in the standard fashion to produce a lookahead tree of height 3 (= $\lceil \log_4 48 \rceil$). The total time to add is thus (3x2)x50=150 nS plus the delay through a full adder (50 nS). The third stage performs normalization using a shift register--normalization after multiply never requires more than a one position right shift if numbers are represented in the format above. The final step in stage three--exponent addition--is performed using a simple 8 bit ripple carry adder. The effect of normalization on the exponent is also accounted for by reusing this adder.

Notice that in both the design of the AU and the MU very conservative timing estimates were used. The only critical parts are stage one of the AU (alignment) and stage one of the MU (the carry-save adder tree).

The PM is to be realized as a 1Kx50 bit dynamic memory. The design is based on the standard single transistor dynamic memory cell (see [CB80]). The memory is organized as 50 "planes" of 32x32 cells. The PM is addressed using a 4x10 bit program counter stack which allows convenient subroutine linkage between subroutines nested up to three deep. Refresh for the memory is achieved by cycle stealing every 16th instruction fetch (this has not been taken into account in the performance figure of the next section). The refresh address is kept in a 5 bit counter that is incremented every 16th M-cycle. The PM can be regarded as being a writeable control store, i.e., programming the NP is essentially done at the microcode level. The instruction format is shown in Figure 3. There are two basic types of instructions distinguished by the leftmost two bits.

Type 1 control the AU and MU indicating which registers in the RF are the sources for their operands and which registers are the destinations for their results. Fields SA1 and SA2 indicate sources for the AU, and field DA indicates a destination for the AU's result. Similarly for the MU--see Figure 3. Provision is made to specify a no-operation for the AU and/or the MU. Since both the AU and the MU are three stage pipelines and since it takes one M-cycle to move data to these

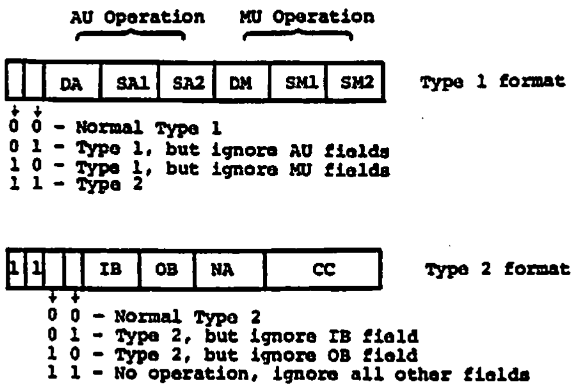

Figure 3. Instruction Formats.

units (see later), the destination field information is not needed by the control logic until four M-cycles after the source field information. To account for this both the destination fields of the PMDR are piped through their own four stage delay lines before being decoded by the control logic. The leftmost two bits must also be piped through a four stage delay to allow the control logic to determine whether or not to ignore the output of the destination field delay lines. This technique for controlling pipelines is explained in more detail in [Ko77].

Type 2 instructions control data transfers from the head of the IB FIFO to registers in the RF, as well as from the registers in the RF to the tail of the OB FIFO (specified by fields IB and OB in the format of Figure 3). Type 2 instructions also handle branching. A 10 bit next address field (NA in the format of Figure 3) is stacked on the PCS if the condition indicated by the CC field is met. Conditions include: IB full; OB full; result of add positive; result of add negative; result of add zero; always true, i.e. an unconditional branch. Detection of the conditions is performed by the condition code logic--CC in Figure 3. To use the NP efficiently type 2 instructions should be kept to a minimum.

Notice that the instruction format is very "horizontal" allowing concurrent operation of the AU and the MU to be specified. The job of taking advantage of this potential for concurrency is left entirely up to the user. This means that program preparation is quite complex if maximum use is to be made of the NP. However, as stated in the introduction the NP is intended for dedicated environments where it is likely to execute only a very small set of programs. The development of these programs should be considered as part of the overall system design. As noted earlier, this approach to program development is more in line with microcode development than standard program development.

The RF is a 256x32 bit static memory. The design is based on the standard six transistor static memory cell (see [OY80]). It is organized as 256 32 bit registers. The registers share a single 32 bit wide output bus and a single 32 wide input bus (see Figure 2). The output bus connects the registers to the two AU inputs, to the two MU inputs, and to the tail of the output buffer, OB. During type 1 instructions the use of the output bus is multiplexed. Data is moved from the RF registers to the two AU inputs and the two MU inputs in four steps--one step per input. The complete transfer takes an M-cycle; each step takes 125 nS. The input bus connects the output of the AU, the output of the MU, and the head of the IB to the RF registers. As with the output bus, during type 1 instructions, the input bus is multiplexed. Data is moved from the AU output and the MU output in two steps, one step per output. The complete transfer takes an M-cycle; each step takes 250 nS.

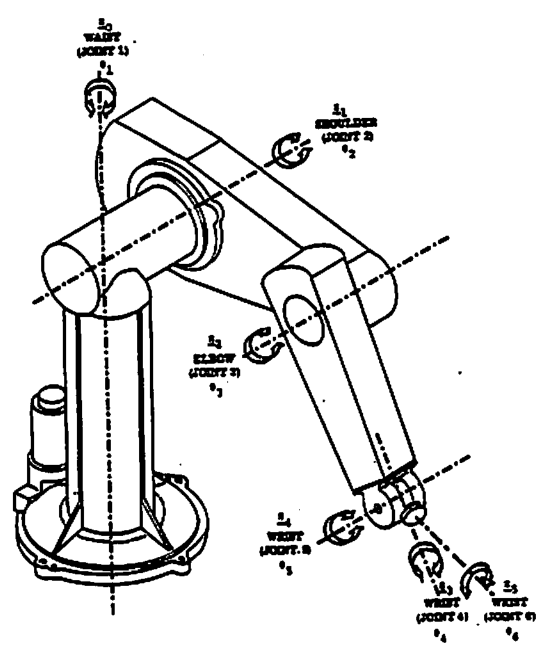

Figure 4. PUMA 600 Robot Arm.

For data to make a round trip from a register through a function unit and back to a register takes five M-cycles.

The IB and the OB are 32 word FIFO buffers for input and output respectively. In the case of the IB, data can be added to the tail and removed from the head asynchronously, unless the buffer is full. Adding to the tail is under the control of an external clock which need not run synchronously with the chip timing. This requires a synchronizer circuit. Designing correctly operating synchronizers can be very involved; however, it need not be since the problem has been thoroughly studied in [SC79]. The operation of the OB is also asynchronous in a similar fashion.

Finally, the PM and the PCS can be loaded through the input port to allow the chip to function as an attached processor.

ROBOT CONTROL

As mentioned earlier, the design philosophy of the NP is oriented toward dedicated numerically intensive applications. A practical example of this type of application is the control of a robot arm where the arm response is limited by the complexity of control computation. In particular, we are interested in using the NP in the real time control of a robot arm, specifically, the Unimation PUMA 600. It is a six link arm having all revolute joints and is illustrated in Figure 4.

The overall control strategy for an arm such as the PUMA involves five basic stages. These stages are as follows.

(1) A path planning stage. The user specifies a sequence of points through which the arm's hand should move. This stage of the strategy determines a cartesian space hand trajectory which passes through the user specified points without exceeding actuator torque limitations [WL78].

(2) Orientation matrix calculation. Orientation matrices A_0^i of the ith link with respect to the base, or zeroth, frame are determined [DH55]. These are used in stages 3 and 4.

(3) Trajectory transformation. The cartesian coordinate hand trajectory given in terms of the angular velocity, $\underline{\omega}_h$ and linear velocity, $\underline{v}_h$, must be converted into equivalent joint angular velocities, $\dot{\underline{\theta}}$ and accelerations $\ddot{\underline{\theta}}$ (six vectors for the PUMA arm). There are several approaches to this problem [Wh69], [LW80a].

(4) Gross motion control. The actuator torques $\underline{\tau}$ (a six vector) required to achieve the previously determined $\dot{\underline{\theta}}$ and $\ddot{\underline{\theta}}$ are computed using an iterative set of equations [LW80b], [TM80]. . Because this stage represents a potential computational bottleneck, we have chosen to benchmark our NP processor using it. The equations are shown in more detail in Figure 5. Where $\underline{\omega}_i$ and $\underline{\alpha}_i$ are angular velocity and acceleration; $\underline{a}_i$ is the linear acceleration; $\underline{f}_i$, and $\underline{n}_i$ the force and torque at the ith joint; $\underline{r}_i$ and $\underline{R}_i$, the distance from the (i-1)th origin to the ith origin and center of mass, respectively. J_i is an inertial tensor about the (i-1)th axis. All vectors and tensors are represented in frames fixed in the ith frame. The actuator torques τ are obtained from the torques $\underline{n}_i$ by $\tau_i = \underline{z}_{i-1} \cdot \underline{n}_i$. See [TM80] for more details.

(5) Fine Motion (Accommodation). To achieve the final hand adjustment, techniques similar to the one described in [PS76] are normally used. We plan to employ a new method that makes use of the same dynamic equations used in the gross motion phase.

$$\underline{\omega}_i = A_i^{i-1} (\underline{\omega}_{i-1} + \dot{\theta}_i \underline{z}_{i-1})$$

$$\underline{\alpha}_i = A_i^{i-1} (\underline{\alpha}_{i-1} + \ddot{\theta}_i \underline{z}_{i-1} + \underline{\omega}_{i-1} \times \underline{z}_{i-1} \dot{\theta}_i)$$

$$\underline{a}_i = \phi_i \underline{r}_i + A_i^{i-1} \underline{a}_{i-1}$$

$$\underline{f}_i = \phi_i m_i \underline{R}_i + A_i^{i-1} m_i \underline{a}_{i-1} + A_i^{i+1} \underline{f}_{i+1}$$

$$\underline{n}_i = \underline{n}_i' + m_i \underline{R}_i \times A_i^{i-1} \underline{a}_{i-1} + \underline{r}_i \times A_i^{i+1} \underline{f}_{i+1} + A_i^{i+1} \underline{n}_{i+1}$$

$(\underline{n}_i')_x = (\phi_i J_i)_{zy} - (\phi_i J_i)_{yz}$ and similarly for y and z

and where

$$\phi_i = \begin{pmatrix} -(\omega_y^2 + \omega_z^2) & \omega_x\omega_y - \alpha_z & \omega_x\omega_z + \alpha_y \\ \omega_y\omega_x + \alpha_z & -(\omega_z^2 + \omega_x^2) & \omega_y\omega_z - \alpha_x \\ \omega_z\omega_x - \alpha_y & \omega_z\omega_y + \alpha_x & -(\omega_x^2 + \omega_y^2) \end{pmatrix}$$

$$J_i = \begin{pmatrix} \frac{-I_{xx}+I_{yy}+I_{zz}}{2} & R_xR_y & R_xR_z \\ R_yR_x & \frac{I_{xx}-I_{yy}+I_{zz}}{2} & R_yR_z \\ R_zR_x & R_zR_y & \frac{I_{xx}+I_{yy}-I_{zz}}{2} \end{pmatrix}$$

where I is the inertial tensor of the ith link about the (i-1)th axis.

Figure 5. Equations of Motion.

SIMULATION RESULTS

To illustrate the effectiveness of the NP a functional simulation was performed using APL. The iterative set of equations for computing the actuator torques were used as a benchmark (see Figure 5). These were programmed for the NP. A listing of the program is given in the Appendix. The NP is operating at its maximum rate when both function units are in streaming mode. In this mode it is producing the results of two floating-point operations every M-cycle, i.e. it is operating at a rate of 4 MFLOPS. Our simulation showed that about 73% of the time the function units produced results, i.e. the NP was operating at an average of 2.93 MFLOPS for this benchmark. To achieve this considerable time was spent hand optimizing the program (see Appendix). Scheduling two pipelined function units is time consuming. Support software to help with this aspect of program preparation is being considered.

CONCLUSION

The encouraging performance figures of the simulation indicate that the NP would be a valuable component for the designer of real-time systems who is faced with compute bound bottlenecks. Further simulation is being performed using the remaining components of the control strategy as benchmarks, and the addition of another function unit to perform reciprocals is being planned. The ultimate success of this study depends on whether or not the detailed circuit design can be carried through successfully to realize the NP as we have specified it.

The authors would like to thank Professors Dan Atkins and George Lee for their advice and comments.

REFERENCES

[CB80] Chan, J. Y., J. J. Barnes, C. Y. Wang, J. M. DeBlasi, and M. R. Guidry, "A 100nS 5V Only 64Kx1 MOS Dynamic RAM," IEEE J. Solid-State Circuits, Vol. SC-15, No. 5, Oct. 1980, pp. 839-846.

[Co80] Coonen, J. T., "An Implementation Guide to a Proposed Standard for Floating-Point Arithmetic," Computer Magazine, Jan. 1980, pp. 68-79.

[DH55] Denavit, J., R. S. Hartenberg, "A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices," Jour. Applied Mechanics, Jun. 1955.

[F179] Processor Handbook, Floating Point Systems, Document #860-7259-003, Feb. 1979.

[In79] The 8086 Family User's Manual, Intel Corp., Oct. 1979.

[Ko77] Kogge, P. M., "The Microprogramming of Pipelined Processor," Proc. of the 4th Annual Symp. on Computer Architecture, March 1977, pp. 63-69.

[Ku78] Kuck, D. J., The Structure of Computers and Computations, John Wiley, 1978.

[LM80] Lee, C. S., T. N. Mudge, E. J. Delp, R. A. Volz, D. E Atkins, and S. Ganapathy, Advanced Control for Multirobot Assembly Systems, Proposal to NSF Automation, Bioengineering, and Sensing Program, Dec. 1980.

[LW80a] Luh, J.Y.S., M. W. Walker and R.P.C. Paul, "Resolved-Acceleration Control of Mechanical Manipulators," IEEE Trans. Automatic Control, Vol. AC-25, No. 3, Jun. 1980, pp. 468-473.

[LW80b] Luh, J.Y.S., M. W. Walker, and R.P.C. Paul, "On-Line Computational Scheme for Mechanical Manipulators," Trans. ASME: Jour. of Dynamic Systems, Measurement, and Control, Vol. 120, Jun. 1980, pp. 69-76.

[MC80] Mead, C. A., and L. A. Conway, Introduction to VLSI Systems, Addison-Wesley, 1980.

[OY80] Ohzone, T., J. Yasui, T. Ishihara, and S. Horiuchi, "An 8Kx8 Bit Static MOS RAM Fabricated by n-MOS/n-Well CMOS Technology," IEEE J. Solid-State Circuits, Vol. SC-15, No. 5, Oct. 1980, pp. 854-861.

[Pa80] Palmer, J., "The Intel 8087 Numeric Data Processor," Proc. 7th Annual Symp. on Computer Architecture, La Baule, France, May 1980, pp. 174-181.

[PS80] Patterson, D. A., and C. H. Sequin, "Design Considerations for Single-Chip Computers of the Future," IEEE Trans. Computers, Vol. C-29, No. 2, Feb. 1980, pp. 108-116.

[PS76] Paul, R., and B. Shimano, "Compliance and Control," Proc. of Joint Automatic Control

Conference, Purdue University, Jul. 1976.

[SC79] Stucki, M. J., and J. R. Cox, "Synchronization Strategies," Proc. Caltech Conf. on VLSI, Jan. 1979.

[TM80] Turney, J., T. N. Mudge, C.S.G. Lee, Equivalence of Two Formulations for Robot Arm Dynamics, SEL Report 142, ECE Department, University of Michigan, Dec. 1980.

[Wh69] Whitney, D. E., "Resolved Motion Rate Control of Manipulators and Human Prostheses," IEEE Trans. Man-Machine Systems, Vol. MMS-10, No. 2, Jun. 1969, pp. 47-53.

[WL78] Walker, M. W., and J.Y.S. Luh, Manipulator Control, Technical Report TR-EE78-12, School of Electrical Engineering, Purdue University, Mar. 1978.

APPENDIX

Below is a partial listing of the program to compute the actuator torques, i.e. one step through the iterative set of equations shown in Figure 5. The program is in symbolic form. The left hand column indicates the program step. The second and third column indicate the activity of the multiplier and adder respectively. Zeroes indicate no activity, i. e. the corresponding function unit is not initiated at that time step. All the instructions are type 1, or type 2 no-operations.

The program was assembled to correspond to the format of Figure 3 with symbolic names being assigned to specific registers in RF. This assembled form of the program was used to drive the functional simulation of the NP.

A few comments are in order about the variables used in the program below. First, θ dot refers to $\dot{\theta}$, and θ dotdot refers to $\ddot{\theta}$. Primed variables and variables followed by A or B are temporary variables. Other variables names are self explanatory. For example, M2 refers to the mass of link 2, R2Z refers to the center of mass coordinate of link 2, and J1XX refers to the xx component of the inertial tensor J for link 1.

PUMA ARM EXAMPLE:

STEP	MULTIPLIER					ADDER				
0	SINθ2	*	θDOT1	→	WX	0	-	0	→	0
1	COSθ2	*	θDOT1	→	WY	0	-	0	→	0
2	SINθ2	*	θDOTDOT1	→	EX	0	-	0	→	0
3	COSθ2	*	θDOTDOT1	→	EY	0	-	0	→	0
4	J1XX+J1ZZ	*	θDOTDOT1	→	N1Y	0	-	0	→	0
5	θDOT2	*	WX	→	EB	0	-	0	→	0
6	θDOT2	*	WY	→	EA	0	+	θDOTDOT2	→	EZ
7	WX	*	WY	→	ΦXY	0	+	θDOT2	→	WZ
8	WX	*	WX	→	ΦXX	0	-	0	→	0
9	WY	*	WY	→	ΦYY	0	-	N1Y	→	N1Y
10	COSθ2	*	G	→	AB	EB	+	EY	→	EY
11	SINθ2	*	G	→	AA	EA	+	EX	→	EX
12	WZ	*	WZ	→	ΦZZ	ΦXY	+	EZ	→	ΦXY
13	WY	*	WZ	→	ΦYZ	ΦXY	-	EZ	→	ΦXY
14	WX	*	WZ	→	ΦXZ	ΦXZ	+	ΦZZ	→	ΦZZ
15	M2*R2Z	*	AB	→	F2X	0	+	WX	→	WX
16	M2*R2X	*	AB	→	F2Z	0	-	WY	→	WY
17	M2*R2Z	*	AA	→	F2Y	ΦXZ	+	ΦYY	→	ΦXZ
18	ΦYX	*	A2	→	AX'	ΦYZ	+	WX	→	ΦXY
19	ΦYX	*	J2XX	→	NZ'	EX	-	ΦYZ	→	ΦYZ
20	ΦYX	*	J2XZ	→	NX'	EY	-	ΦXZ	→	ΦXZ
21	ΦZZ	*	D2	→	AZ'	EY	+	ΦXZ	→	ΦZX
22	ΦXX	*	A2	→	AX'	0	-	EX	→	EX
23	ΦXX	*	J2XZ	→	NY'	AX'	-	AB	→	AX'
24	ΦYZ	*	D2	→	AY	NZ'	-	N2Z	→	N2Z
25	ΦXZ	*	D2	→	AX	N2X	-	NX'	→	N2X
26	ΦZX	*	A2	→	AZ	0	-	AZ'	→	AZ'
27	ΦXZ	*	J2ZZ	→	NY'	AX'	+	AA	→	AX'
28	ΦZY	*	J2YY	→	NX'	N2Y	+	NY'	→	N2Y
29	ΦYZ	*	J2XZ	→	NZ'	AY	+	AY'	→	AY
30	ΦZX	*	J2XX	→	NY'	0	-	0	→	0
31	ΦYZ	*	J2ZZ	→	NX'	AZ'	-	AZ	→	AZ
32	ΦZZ	*	J2ZZ	→	NA	AX	-	AX'	→	AX
33	ΦXY	*	J2YY	→	NZ'	NY'	-	N2Y	→	N2Y
34	θDOT3	*	WY	→	WA	N2X	-	NX'	→	N2X
35	θDOT3	*	WX	→	WB	N2Z	+	NZ'	→	N2Z
36	COSθ3	*	WX	→	WA	0	-	0	→	0
37	SINθ3	*	WY	→	WA'	0	-	0	→	0
38	SINθ3	*	EX	→	EB	N2Y	+	NY'	→	N2Y
39	COSθ3	*	EY	→	EB'	EY	+	EA	→	EX
40	COSθ3	*	AX	→	AA	EY	-	EB	→	EY
41	SINθ3	*	AY	→	AA'	N2X	-	NX'	→	N2X
42	SINθ3	*	AX	→	AB	WA	+	WA'	→	WX
43	COSθ3	*	AY	→	AB'	WZ	+	θDOT3	→	WY
44	COSθ3	*	EZ	→	EA	WB	-	WB'	→	WZ
45	SINθ3	*	EY	→	EA'	N2Z	-	NZ'	→	N2Z
46	SINθ3	*	EX	→	EB	AA	+	AA'	→	AX
47	COSθ3	*	EY	→	EB'	0	+	AZ	→	AY
48	EZ	*	WY	→	ΦZY	AB	-	AB'	→	AZ
49	WX	*	WZ	→	ΦXZ	EZ	+	θDOTDOT3	→	EZ
50	WY	*	WZ	→	ΦYZ	EA	+	EA'	→	EX
51	WY	*	WY	→	ΦXX	N2Y	+	NA	→	N2Y
52	WX	*	WX	→	ΦXX	EB	-	EB'	→	EZ
53	WZ	*	WZ	→	ΦYY	0	-	0	→	0
54	M3	*	AX	→	F3X	ΦYZ	+	EY	→	ΦYZ
55	M3	*	AY	→	F3Y	ΦYZ	-	EX	→	ΦYZ
56	M3	*	AZ	→	F3Z	ΦYZ	+	EZ	→	ΦZY
57	M3*R3Z	*	AY	→	N3X	ΦXX	+	ΦZZ	→	ΦZZ
58	M3*R3Z	*	AX	→	N3Y	ΦXZ	-	EY	→	ΦZX
59	ΦXZ	*	M3*R3Z	→	FX'	ΦXY	+	EZ	→	ΦYX
60	ΦYZ	*	M3*R3Z	→	FY'	ΦZY	-	EZ	→	ΦXY
61	ΦZY	*	J3YY	→	NX'	0	-	0	→	0
62	ΦZZ	*	M3*R3Z	→	FZ'	0	-	0	→	0
63	ΦXZ	*	J3ZZ	→	NY'	0	-	0	→	0
64	ΦYX	*	J3XX	→	N3Z	F3X	+	FX'	→	F3X
65	ΦYZ	*	J3ZZ	→	NX'	F3Y	+	FY'	→	F3Y
66	ΦZX	*	J3XX	→	NY'	NX'	-	N3X	→	N3X
67	ΦXY	*	J3YY	→	NZ'	F3Z	-	FZ'	→	F3Z
68	θDOT4	*	WY	→	WA	NY'	+	N3Y	→	N3Y
69	θDOT4	*	WX	→	WB	0	-	0	→	0
70	COSθ4	*	WY	→	WA	0	-	0	→	0
71	SINθ4	*	WY	→	WA'	N3X	-	NX'	→	N3X
72	SINθ4	*	WX	→	WB	N3Z	-	NZ'	→	N3Z
73	COSθ4	*	WY	→	WB'	EX	+	EA	→	EX
74	COSθ4	*	AX	→	AA	EY	-	EB	→	EY
75	SINθ4	*	AY	→	AA'	N3Y	-	NY'	→	N3Y
76	SINθ4	*	AX	→	AB	WA	+	WA'	→	WX
77	COSθ4	*	AY	→	AB'	WZ	+	θDOT4	→	WY
78	COSθ4	*	EX	→	EA	WB'	-	WB	→	WZ
79	SINθ4	*	EY	→	EA'	0	-	AZ	→	AY
80	SINθ4	*	EX	→	EB	AA	+	AA'	→	AA
81	COSθ4	*	EY	→	EB'	EZ	+	θDOTDOT4	→	EY
82	WX	*	WY	→	ΦXY	AB'	-	AB	→	AB
83	WX	*	WZ	→	ΦXZ	0	-	0	→	0
84	WY	*	WZ	→	ΦYZ	EA	+	EA'	→	EX
85	WX	*	WX	→	ΦZZ	0	-	WY	→	WY
86	WZ	*	WZ	→	ΦYY	EB'	-	EB	→	EZ
87	M4	*	AA	→	F4X	0	-	EY	→	EY
88	M4	*	AY	→	F4Y	ΦYZ	+	EY	→	ΦZX
89	M4	*	AB	→	F4Z	ΦYZ	-	EX	→	ΦZY
90	M4*R4Y	*	AB	→	N4X	ΦXZ	-	EY	→	ΦXZ
91	M4*R4Y	*	AA	→	N4Z	ΦZY	+	EZ	→	ΦXY
92	0	*	0	→	0	ΦYY	+	ΦZZ	→	ΦYY
93	ΦZX	*	J4XX	→	NY'	ΦXY	-	EZ	→	ΦYX
94	ΦZY	*	D4	→	AZ	ΦYZ	+	EX	→	ΦYZ
95	M4*R4Y	*	ΦZY	→	FZ'	0	-	0	→	0
96	ΦXY	*	D4	→	AX	0	-	0	→	0
97	ΦZY	*	D4	→	AY'	0	-	0	→	0
98	M4*R4Y	*	ΦXY	→	FX'	0	-	0	→	0
99	M4*R4Y	*	ΦYY	→	FY'	AB	-	AZ	→	AZ
100	ΦZY	*	J4YY	→	NX'	F4Z	-	FZ'	→	F4Z
101	ΦYX	*	J4XX	→	NZ'	AA	-	AX	→	AX
102	ΦXZ	*	J4ZZ	→	N4Y	AY	-	AY'	→	AY
103	ΦYZ	*	J4ZZ	→	NY'	F4X	-	FX'	→	F4X
104	ΦXY	*	J4YY	→	NZ'	FY'	-	F4Y	→	F4Y
105	θDOT5	*	WY	→	EA	N4X	-	NX'	→	N4X
106	θDOT5	*	WX	→	EA'	NZ'	+	N4Z	→	N4Z