References

[1] Corradini, M.L., Kim, B.J., and Oh, M.D. Vapor Explosions in Light Water Reactors: A Review of Theory and Modeling. **Progress in Nuclear Energy.** 22, No.1, (1988): 1-117.

[2] Garland, F. and Atkinson, G. The Interaction of Liquid Hydrocarbon with Water. **Final Report on Contract No. DOT-CG-11,911 A.** U.S.Coast Guard, Washington, USA, October 1971.

[3] Burgess, D.S., Murphy, J.N., and Zabetakis, M.G. Hazards of LNG Spillage in Marine Transportation. **Final Report No. S-4105.** U.S. Departmenet of the Interior, Bureau of Mines, Pittsburgh, Pennsylvania, February 1970.

[4] Sallack, J.A.. An Investigation of Explosion in the Soda Smelt Dissolving Operations. **Pulp and Paper Magazine of Canada.** 56 (1955):114-118.

[5] Duckworth, R.C., Murphy, J.G., Utschig, T.T., Corradini, M.L., Merrill, B.J., and Moore, R.L. Analysis of Liquid Cryogen-Water Experiments with the MELCOR Code. **ANS 14th Topical Meeting on the Technology of Fusion Energy.** October, 2000.

[6] Takahashi, T.J. Explosion as Lava Enters Ocean at Kilauea, Hawaii. http://geopubs.wr.usgs.gov, USGS.

[7] Long, G. Explosion of Molten Aluminum in Water- Cause and Prevention. **Metal Progress.** 71(May 1957):107-112.

[8] Buxton, L.D. and Nelson, L.S. Core-Meltdown Experimental Review. **SAND 74-0382.** August, 1975.

[9] Bang, K.H. A Study of Stratified Vapor Explosions. **Ph.D. Thesis.** University of Wisconsin-Madison, Wisconsin, 1989.

[10] Meyer, L., Schumacher, G. and Jacobs, H. Investigation of the Premixing Phase of a Steam Explosion with Hot Spheres. **Nuclear Technology.** 123 (August 1998):142-155.

[11] Magallon, D., Hohmann, H., and Schins, H. Pouring of 100-kg-Scale Molten UO2 into Sodium. **Nuclear Technology.** 98(April 1992):79-90.

[12] Sunchai, N. and Urith A. Installation for Low Temperature Vapor Explosion Experiment. **Proceedings of the Workshop on Severe Accident Research.** Japan. November, 2000.

[13] Sunchai, N. and Urith A. Low Temperature Vapor Explosion**. Research Report No. 153-NC-2541,** Engineering Institute for Research and Development, Faculty of Engineering, Chulalongkorn University, Thailand, August 2000.

[14] Urith, A., Sunchai, N., and Tatchai, S. Investigation on the Possibility of the Low Temperature Vapor Explosion. **Third Korea-Japan Symposium on Nuclear Thermal Hydraulics and Safety.** Kyeongju, Korea, October 2002.

[15] Witte, L.C. and Vyas, T.J., Gelabert, A.A. Heat Transfer and Fragmentation during Molten Metal/Water Interaction. **J. Heat Transfer.** 95 (1973):521-527.

[16] Dullforce, T.A., Buchanan, D.J., and Peckover, R.S. Self-Triggering of Small-Scale Fuel-Coolant Interaction: I. Experiments. **J. Physics D:Applied Physics.** 9(1976):1295-1303.

[17] Zyszkowski, W. Thermal Interaction of Molten Copper with Water. **Int. J. Heat Mass Transfer.** 18(1975):271-287.

[18] Nelson, L.S. and Duda, P.M. Steam Explosions Experiments with Single Drops of Iron Oxide Melted with CO2 Laser. **SNL, SAND81-1346, NUREG/CR-2295**, September 1981.

[19] Nelson, L.S. and Duda, P.M. Steam Explosions Experiments with Single Drops of Iron Oxide Melted with CO2 Laser; Part 2. Parametric Studies. **SNL, SAND82-1105, NUREG/CR-2718**, 1984.

[20] Abe, Y., Nariai, H., and Hamada, Y. The Trigger Mechanism of Vapor Explosion. **J. Nuclear Science and Technology.** 39 No.8 (August 2002):845-853.

[21] Henry, R.E. and Fauske, H.K. Required Initial Conditions for Energetic Steam Explosions. **ASME HTD-V19.** Washington, D.C., 1981.

[22] Corradini, M.L. and Moses, G.A. A Dynamic Model of Fuel-Coolant Mixing. **Proc. Int. Mtg. Light Water Reactor Severe Accident Evaluation.** Cambridge, MA. 1(1983):63

[23] Mitchell D.E. and Evans, N.A. The Effect of Water and Fuel Mass Ratio and Geometry on the Behavior of Molten Core-Coolant Interaction at Intermediate Scale. **Proc. Int. Mtg. Thermal Nuclear Reactor Safety.** Chicago, IL, NUREG/CP-0027.

[24] Theofanous, T.G. and Saito, M. An Assessment of Class-9 (Core-Melt) Accidents for PWR Dry Containment System. **Nuclear Engineering Design.** 66(1982):307.

[25] Corradini, M.L., Murphy, J., Nilsuwankosit, S., Pineau, D., Shamoun, B., Tang, J., and Wang, S. **Fuel-Coolant Interaction Analyses with TEXAS-V Vapor Explosion Model.** Nuclear Safety Research Center, University of Wisconsin-Madison, Wisconsin, USA, 1998.

[26] Tang, J. Modeling of the Complete Process of One-Dimensional Vapor Explosions. **Ph.D Thesis.** University of Wisconsin-Madison, Wisconsin, USA. 1993.

[27] Silbey, R.J. and Alberty, **R.A. Physical Chemistry.** (n.p.):John Wiley & Sons, 2001.

[28] Cronenberg, A.W. and Benz, R. Vapor Explosion Phenomena with Respect to Nuclear Reactor Safety Assessment. **NUREG/CR-0245**, 1978.

[29] Kim, B.J. Heat Transfer and Fluid Flow Aspects of a Small-Scale Single Droplet Fuel-Coolant Interactions. **Ph.D. Thesis.** University of Wisconsin-Madison, Wisconsin, USA. 1985.

[30] Board, S.J. and Hall, R.W. Propagation of Thermal Explosions. 2-Theoretical Model. **CEGB Report RD/B/N 3249**, 1974.

[31] Fox, R.W. and McDonald, A.T. **Introduction to Fluid Mechanics.** (n.p.):John Wiley & Sons, 5th edition, 1998.

[32] Shamoun, B.I. and Corradini, M.L. Supercritical Vapor Explosions: Comparisons between Thermodynamic and Mechanistic Models. **Nuclear Technology.** 120(November 1997):158-170.

[33] Fletcher, N.H. **The Chemical Physics of Ice.** Cambridge University Press. 1970.

[34] Hindmarsh, J.P., Russell, A.B., and Chen, X.D. Experimental and Numerical Analysis of the Temperature Transition of a Suspended Freezing Water Droplet. Int. J. Heat and Mass Transfer. 46(2003):1199-1213.

[35] Japanese Standards Association. **JIS Handbook, Piping.** JIS. 1994.

[36] Klein, S.A. and Alvarado, F.L. **EES, Engineering Equation Solver.** F-Chart Software, USA. 1993.

[37] Ray E.B. and George L.T. **CRC Handbook of Tables for Applied Engineering Science.** USA:CRC Press, 1973.

[38] Meeks, M.K., Baker, M.C., and Bonazza, R. Suppression of Vapor Explosions in a Water-Molten-Tin System by Augmentation of the Void Fraction. **Nuclear Technology.** 129(January 2000)69-81.

[39] Wallis, G.B. **One-dimensional Two-phase Flow.** (n.p.):McGraw Hill Book, 1969.

[40] Liu, J., Koshizuka, S., and Oka, Y. Investigation on Energetics of Ex-vessel Vapor Explosion Based on Spontaneous Nucleation Fragmentation. J. of Nuclear Science and Technology. 39(January 2002):31-39.

[41] Chu, C.C. One-Dimensional Transient Fluid Model for Fuel-Coolant Interaction Analysis. **Ph.D. Thesis.** University of Wisconsin-Madison, Wisconsin, USA. 1986.

[42] Epstein, M. and Fauske, H.K. Steam Film Instability and the Mixing of Core Melt Jets and Water. **Proceedings of National Heat Transfer Conference,** Denver, Colorado, USA. 1985.

[43] Uludogan, A. and Corradini, M.L. Modeling of Molten Metal/Water Interactions. **Nuclear Technology.** 109(August 1994):171-186.

[44] Holman, J.P. **Heat Transfer.** (n.p.):McGraw-Hill, 8th edition, 1997.

APPENDIX A

VERY FINE TIME RESOLUTION AND LARGE MEMORY BLOCK MANAGEMENT FOR DATA SAMPLING

A.1 Introduction

The very fine time resolution and the very large size of memory allocation are necessary for measuring and recording the signals in many experiments such as the detection of the pressure signal in the vapor explosion experiment. In general, this can be done with a high-speed oscilloscope with the memory option. Such the oscilloscope is, however, very expensive. In order to reduce the cost and to make use of the available resource, an old PC with Intel 80486 chip and an analog to digital converter (ADC) card are utilized with a freely downloaded C compiler and the PC timing knowledge.

The computer with Intel 80486 chip is an old personnel computer (PC-486), which is available in our laboratory. This computer will be installed with the freely downloaded OpenDOS, for booting up the system, and a 32-bit free C compiler. Both softwares can be downloaded from www.delorie.com [A.1].

The ADC card digitizes the analog signals from the transducers, which can be recorded and analyzed by the computer. The resolution of the signals depends on the digital bits. The more number of bits is available; the better is the signal resolution. In this paper, however, the ADC will not be discussed. Rather, the concern is mainly focused on the timing program using C-language.

DJGPP, a free 32-bit C-language development system for DOS by DJ Delorie, is used in programming the time measurement and the signal-time record. The advantage of DJGPP is not only due to its being freeware but also due to its 32-bit memory addressing capability, since it is based on GCC for UNIX. This gives the user the ability of using the memory over the 640kB block. The developed program requires the functions such as *malloc* and *free* (for allocating and de-allocating the memory), *inportb* and *outportb* (for reading and writing a single byte to and from an 8-bit I/O port), and *fopen* and *fclose* (for opening and closing a file), etc.

The timing on a PC was based on the document written by Kris Heidenstrom (http://home.clear.net.nz/pages/kheidens) [A.2]. The document describes many techniques for timing. It also provides the sample functions and programs for testing. In this paper, we will discuss only the appropriate technique that is used in our experiments. This technique reads the absolute time stamp from the counter/timer chip (CTC) at channel 0 mode three, which has the resolution of 0.8381 μs.

With this specific technique, the very fine measuring and recording system can be developed in a laboratory at a low cost. This should be beneficial to any laboratory measurement, especially the measurement that requires the very fine time resolution and the very large size of memory.

The author would like to express his appreciation to the following persons for their help and contributions, either knowingly or unknowingly, to this writing;

1. Mr. Delorie, for his free 32-bit C compiler,
2. Mr. Kris Heidenstrom, for his article in PC timing,
3. Dr. Sunchai Nilsuwankosit for a PC-486 computer, an Analog to Digital Converter card, and his recommendation on using DJGPP software, and
4. Dr. Arporn Teeramongkonrasmee, my friend, for his recommendation on implementing the PC timing.

A.2 Objectives

1. To utilize a PC-486 as a recording machine.
2. To record a large number of data sets to memory by using a program written with DJGPP.
3. To record the signal at the resolution of 0.8381 μs.

A.3 Basic Information and Programming

<u>Counter/Timer Chip (CTC)</u>

The counter/timer chip (CTC) in the IBM PC family is an Intel 8253 in PC and XT series and an Intel 8254 in AT and later series (see more detail in [2], [3]). This chip sends each signal as an IRQ0 interrupt (the timer tick) to the primary 8259 programmable interrupt controller (PIC) chip as shown in Fig. A.1.

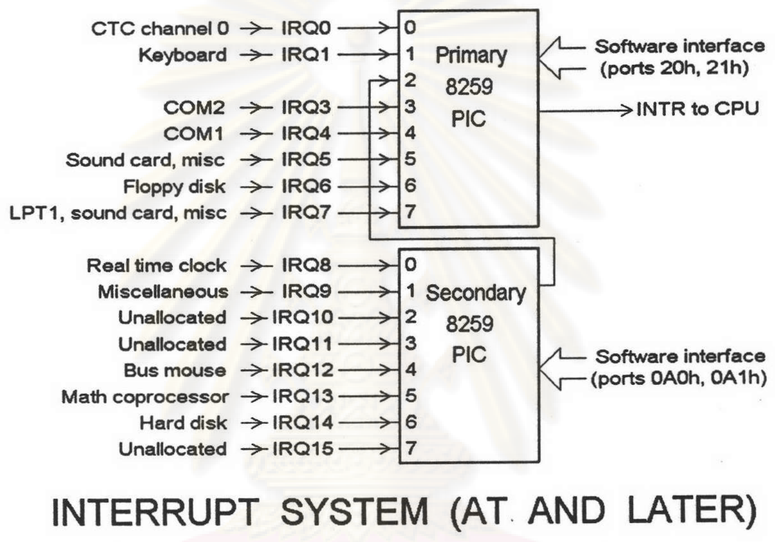


Fig. A.1 Interrupt system in AT and later personnel computer [2]

This IRQ0 interrupt signal, like the other interrupt signals, suspends the process in the processor and causes the processor to execute a special section of code (the interrupt handler). The controller gives IRQ0 the highest priority down to IRQ7, which is the lowest. In addition, IRQ8 to IRQ15 are handled by IRQ2, which lies between IRQ1 (the keyboard scancode interrupt) and IRQ3 (normally used for COM2). This priority is determined by the control bytes sent to PICs by BIOS initialization code.

In Fig. A.2, CTC has three fully independent channels, numbered as zero, one, and two. Each has a clock input, a gate input, and an output. For the purpose of timing, only channel 0 is discussed. CTC channel 0 has the clock input frequency of 1.193182 MHz. Its clock input, gate input and the output to IRQ0 are as shown in Fig. A.3.

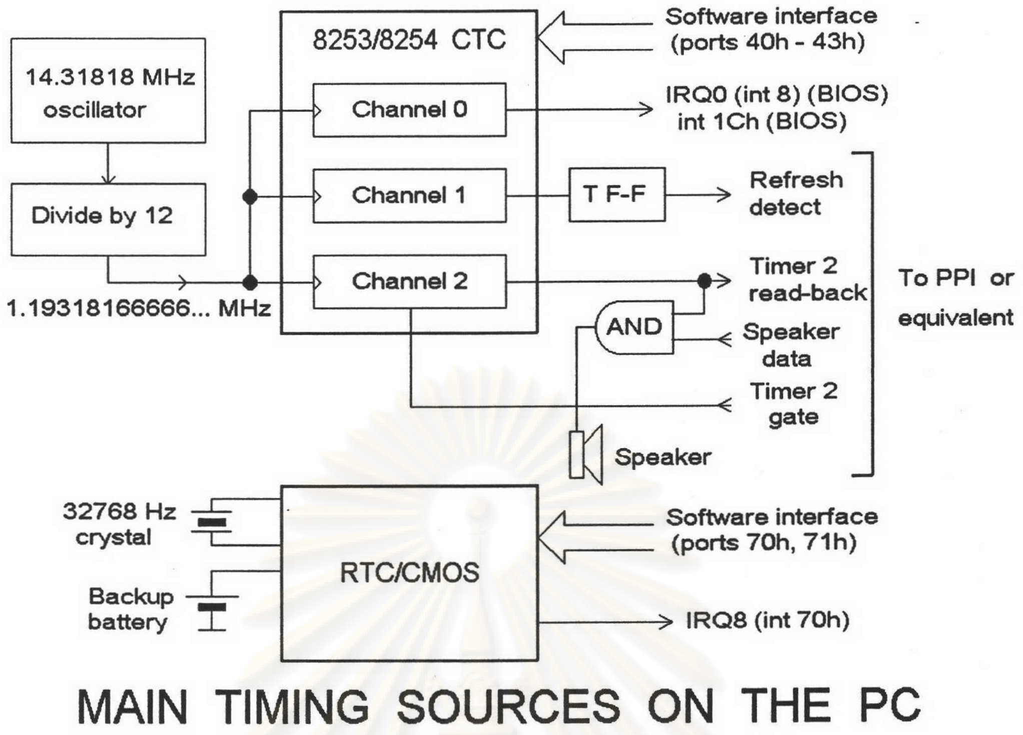


Fig. A.2 Main timing sources on the PC [2]

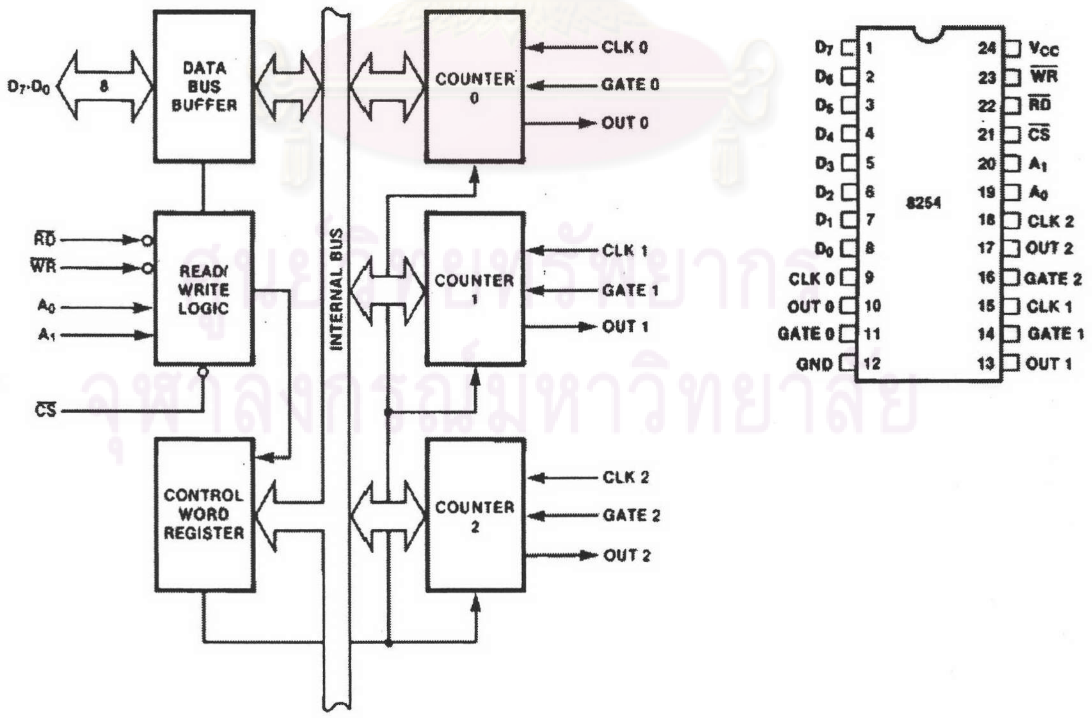


Fig. A.3 CTC block diagram and its pin configuration [3]

The frequency of 1.193182 MHz is divided in channel 0 by the 16-bit counter 0 to produce a lower frequency at the output pin. The divisor, a 16-bit unsigned number between 1 and 65536 (65536 is represented as zero), gives the lowest output frequency, 18.206507364909 Hz (cycle period is 54.92541649846559 ms).

CTC channel 0 or counter 0

Since all three counters are identical in operation, only counter 0 will be described. The internal block diagram of counter 0 is shown in Fig. A.4. The control word register is shown in the same Fig.. While it is not part of the counter itself, its contents determine how the counter operates.

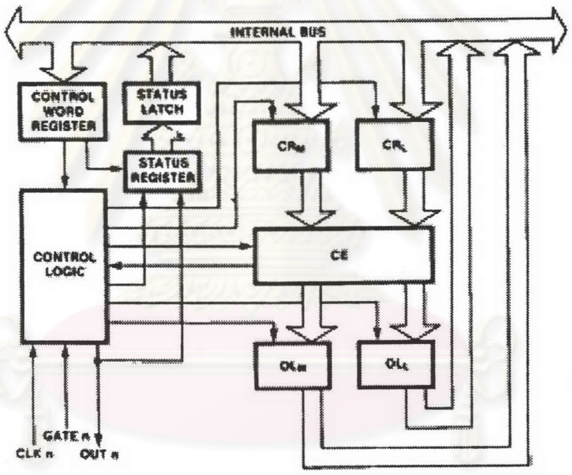


Fig. A.4 Internal block diagram of a counter [3]

The counter is composed of the status register, the status latch, the control logic, the actual 16-bit counting element (CE), the 8-bit output latch of the most significant byte (OL_M), the 8-bit output latch of the least significant byte (OL_L), the 8-bit counter register of the most significant byte (CR_M), and the 8-bit counter register of the least significant byte (CR_L).

OL_M and OL_L are generally called just OL. The OL normally follows the counting value in the CE. However, if a suitable counter latch command is sent to CTC, the OL will latch the present value until it is read by the CPU before returning again to follow the CE. Note that the CE itself cannot be read. Whenever the count value is read, it is the OL that is used.

Like the OL, CR is referred for both CR_M and CR_L. When a new count value is written to the counter, the count value is stored in the CR and later transferred to the CE. The control logic only allows one register at a time to be loaded from the internal bus.

The microcomputer and the CTC device system interface

In this paper, the program reads the count value from the counter 0. To read the count value in the counter, the program has to access the counter ***as the device of the computer***.

The 8254/8253-timer device is a component of a microcomputer system. The software treats it as an array of the peripheral I/O ports or the I/O addresses in the directly addressable I/O page as shown in Table A.1. The system interface between the microcomputer and the CTC device is also shown in Fig. A.5.

Table A.1 Addresses of the counters

Computer I/O Address	Device (Fig.3) Address (A_1A_0)	Computer Description	Device Description	Function
40h	00	Channel 0 data port	Counter 0	Read/Write
41h	01	Channel 1 data port	Counter 1	Read/Write
42h	10	Channel 2 Data port	Counter 2	Read/Write
43h	11	Mode/Command Register	Control Word Register	Write only, Read is ignored

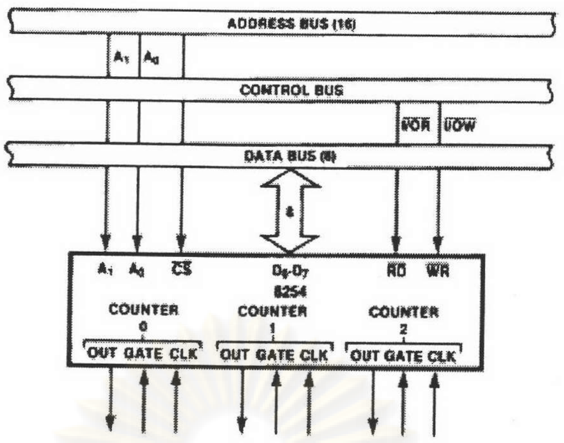


Fig. A.5 CTC 8254 System Interface

Before accessing CTC, a program should disable the computer interrupts during the sequence of accessing to the counter device. This is to ensure that an interrupt service routine does not come along to access the same device during the accessing sequence of the program, which can disrupt the accessing sequence.

To disable the interrupts, the following command is used.

```
disable();
```

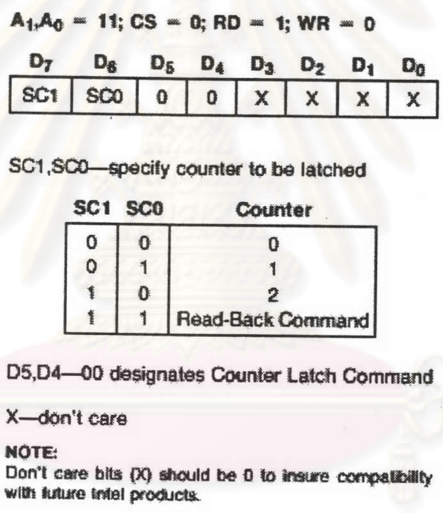
<u>Reading the count</u>

Once the interrupts are disabled, the program can access the CTC device. Before accessing the device, it is often desirable to read the value of a counter without disturbing the count in progress. In the 8254 device, there are three possible methods for reading the counters:

1. the simple read operation,
2. the counter latch command and
3. the read-back command.

The first method is to perform a simple read operation. To read the counter, which is selected by the A_1A_0 inputs, the CLK input of the selected counter as shown in Fig. A.4 must be inhibited by using either the gate input or the external logic. Otherwise, the count value may be in the process of changing when it is read, which gives an undefined result.

The second method uses the counter latch command as shown in Fig. A.6. This command from the computer system data bus is written to the control word register, which is selected when $A_1A_0 = 11$ (43h computer address). The SC0 and the SC1 bits select one of the three counters. The other two bits, D5 and D4, distinguish this command from the other control word commands.



$A_1, A_0 = 11$; CS = 0; RD = 1; WR = 0

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC1	SC0	0	0	X	X	X	X

SC1,SC0—specify counter to be latched

SC1	SC0	Counter
0	0	0
0	1	1
1	0	2
1	1	Read-Back Command

D5,D4—00 designates Counter Latch Command

X—don't care

NOTE:
Don't care bits (X) should be 0 to insure compatibility with future Intel products.

Fig. A.6 Counter Latching Command Format

The selected counter's output latch (OL) gets the count at the time that the command is received. This count value is held in the OL until the CPU has read it (or until the counter is reprogrammed). The count in the OL is then unlatched automatically and the OL returns to follow the counting element (CE). This allows the "on the fly" reading of the counters without affecting the counting in progress.

For the above two methods, the count must be read according to the initial CTC counting format (one-byte counting or two-byte counting).

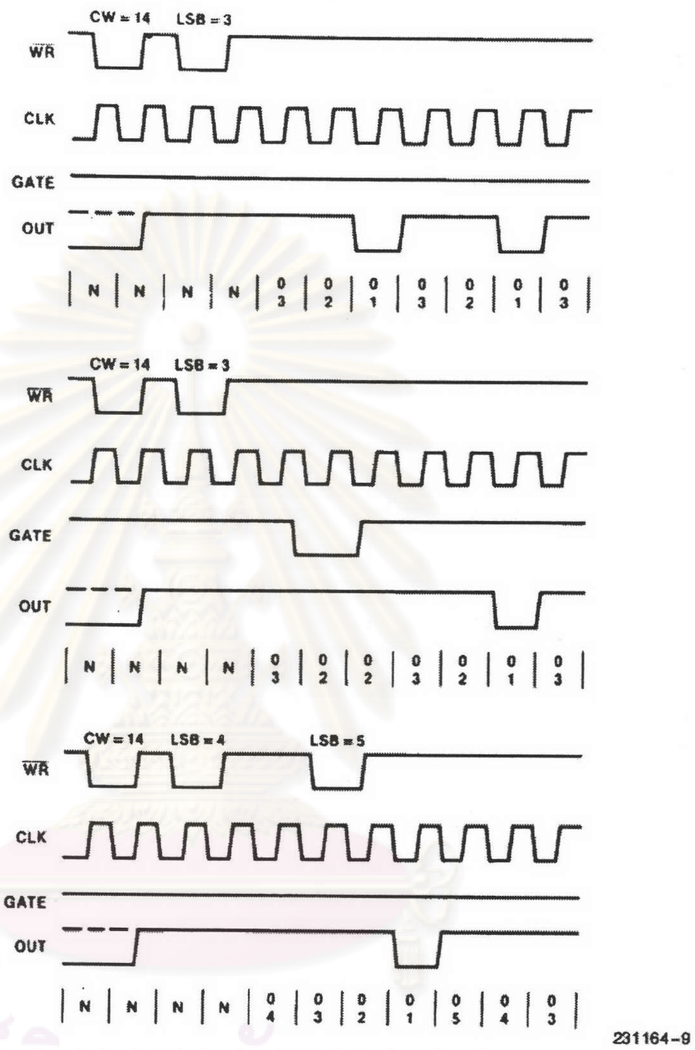
If the counting format is not known, the third method, the read-back command (Fig. A.6), must be used. This command allows the user to check the count value, the programmed mode and the current states of the OUT pin and Null Count flag of the selected counter(s), which is initialized by the past command for counting format. For example, in mode 2 counting with 65536 divisor, the sequence is 0, 65535, 65534, ... 2, 1, 0, 65535, ... and so on. In mode 3, the counting is always an even value, because the counting element is decreased in step of two instead of one. In this case, the count register counts down from the divisor value, in steps of two. When it reaches the value of two, it reloads to the divisor value on the next CTC clock and the output latch toggles its state at this moment. For example, if the divisor is 65536, the counting sequence will be 0, 65534, 65532, … 4, 2, 0, 65534, … and the output pin switches from low to high or from high to low between the cycles.

The example of the timing diagram in mode 2 with the divisor of 3 is shown in Fig. A.7 while Fig. A.8 shows that of mode 3 with the divisor of 4.

Like the second method, the command is written to the control word register with the format as shown in Fig. A.6 and A.7. The command is applied to the counters selected by setting their corresponding bits D3, D2 and/or D1 = 1. As shown in Fig. A.9, D4 is set to 0 to if the status information of the selected counter(s) is needed. If D4 is set to 1, the status information of the selected counter(s) will not be latched. To latch the count value of the selected counter(s), D5 must be set to 0. If it is set to 1, the count value of the selected counter(s) will not be latched.

In this case, the needed commands are as given below.

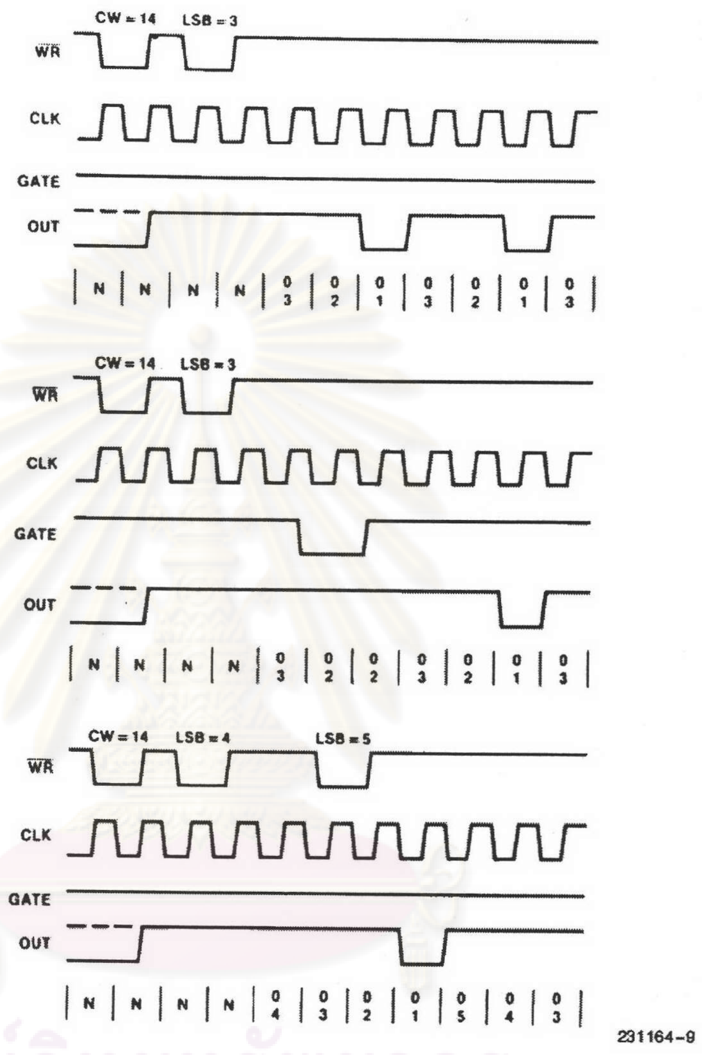
```
outportb(0x43,0xE2);
st1 = inportb(0x40);
```



CW = 14
LSB = 3
WR
CLK
GATE
OUT
N N N N 0 3 0 2 0 1 0 3 0 2 0 1 0 3
CW = 14
LSB = 3
WR
CLK
GATE
OUT
N N N N 0 3 0 2 0 2 0 3 0 2 0 1 0 3
CW = 14
LSB = 4
LSB = 5
WR
CLK
GATE
OUT
N N N N 0 4 0 3 0 2 0 1 0 5 0 4 0 3
231164-9

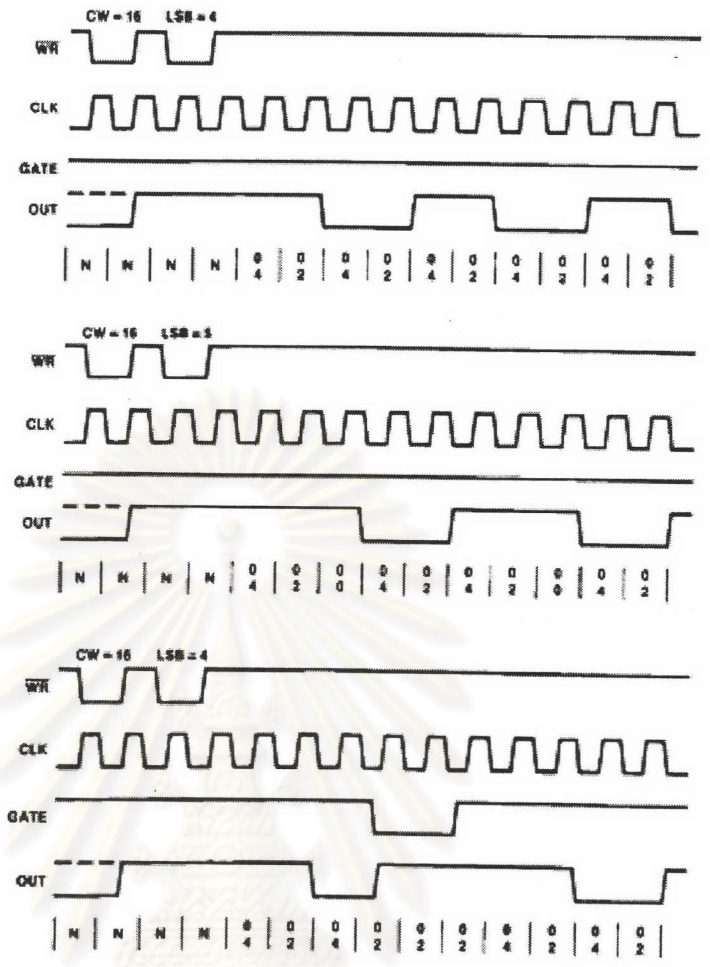
NOTE:
A GATE transition should not occur one clock prior to terminal count.

Fig. A.7 Mode 2 counting


CW = 14
LSB = 3
WR
CLK
GATE
OUT
N N N N 0 3 0 2 0 1 0 3 0 2 0 1 0 3
CW = 14
LSB = 3
WR
CLK
GATE
OUT
N N N N 0 3 0 2 0 2 0 3 0 2 0 1 0 3
CW = 14
LSB = 4
LSB = 5
WR
CLK
GATE
OUT
N N N N 0 4 0 3 0 2 0 1 0 5 0 4 0 3
231164-9

NOTE:
A GATE transition should not occur one clock prior to terminal count.

Fig. A.7 Mode 2 counting



NOTE:
A GATE transition should not occur one clock prior to terminal count.

Fig. A.8 Mode 3 counting

A0, A1 = 11 $\overline{CS}$ = 0 $\overline{RD}$ = 1 $\overline{WR}$ = 0

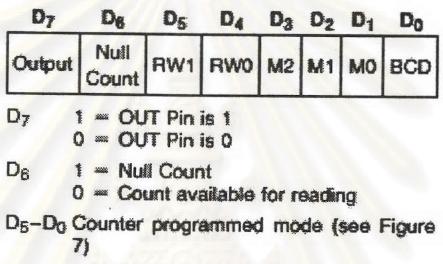
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	1	$\overline{\text{COUNT}}$	$\overline{\text{STATUS}}$	CNT 2	CNT 1	CNT 0	0

D_5: 0 — Latch count of selected counter(s)
D_4: 0 — Latch status of selected counters(s)
D_3: 1 — Select Counter 2
D_2: 1 — Select Counter 1
D_1: 1 — Select Counter 0
D_0: Reserved for future expansion; Must be 0

Fig. A.9 Read-Back Command Format

"`outportb(0x43,0xE2)`" writes the command into the control word register at address *0x43* with command format *E2*. The binary form of this command is 11100010. This command latches the status byte of the counter 0. "`st1 = inportb (0x40)`" reads the status byte from the counter 0 and put the status value into the variable *st1*.

The status format is shown in Fig. A.10. D5 through D0 contain the counter's initialized mode, exactly as written in the last mode control word. Output bit D7 contains the current state of OUT pin.



D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Output	Null Count	RW1	RW0	M2	M1	M0	BCD

D_7	1 = OUT Pin is 1 0 = OUT Pin is 0
D_6	1 = Null Count 0 = Count available for reading
D_5–D_0	Counter programmed mode (see Figure 7)

Fig. A.10 Status Byte

Bits D5 and D4 specify read/write format, as shown in Table A.2. Bits D3, D2 and D1 specify mode of counting, as shown in Table A.3. D0 specify type of counting, BCD or the 16-bit count, as shown in Table A.4. This status format allows the user to monitor the counter's output via software.

Table A.2 RW-Read/Write

D5 (RW1)	D4 (RW0)	Description
0	0	Counter Latch Command (only for read operations)
0	1	Read/Write least significant byte only
1	0	Read/Write most significant byte only
1	1	Read/Write least significant byte first, then most significant byte

Table A.3 M-Mode

D3 (M2)	D2 (M1)	D1 (M0)	Description
0	0	0	Mode 0
0	0	1	Mode 1
X	1	0	Mode 2
X	1	1	Mode 3
1	0	0	Mode 4
1	0	1	Mode 5

X = "don't care bits" should be zero.

Table A.4 BCD

D0 (BCD)	Description
0	Binary Counter 16 bits
1	Binary Coded Decimal (BCD) Counter (4 Decades)

Traditionally, CTC channel zero is set to operate in mode 3 by the BIOS POST. However, some 486 systems may use mode 2 as their defaults. One significant difference between these two systems is the width of the pulse from the CTC pin that triggers the interrupt for the timer tick. This width is narrower in mode 2 but is still wide enough for Intel 8259 PIC chip. The other difference is the value read from the CTC channel zero, which decreases twice as fast in mode 3. As a result, this mode checking is skipped from the variable st1. However, if the counting mode must be ensured, variable st1 must be used.

After reading the status byte, the output bit information is stored in D7 of the variable st1. The counter 0 is then latched. The least significant byte is read first to the variable cv, followed by most significant byte. This is accomplished by the following commands.

```
outportb(0x43,0x00);
cv = inportb(0x40);
cv += inportb(0x40) << 8;
```

The counting value is then converted into up-count, 0-32767.

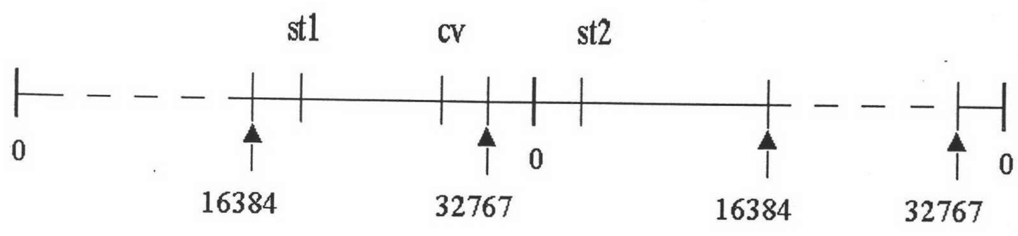
```
cv = ((unsigned short) (0-cv)) >> 1;
```

After receiving the counting value, latch and read the status byte again to ensure that the variable st1 and the variable cv are obtained at the same period, as graphically shown in Fig.11a and 11b. This is done with the following commands.

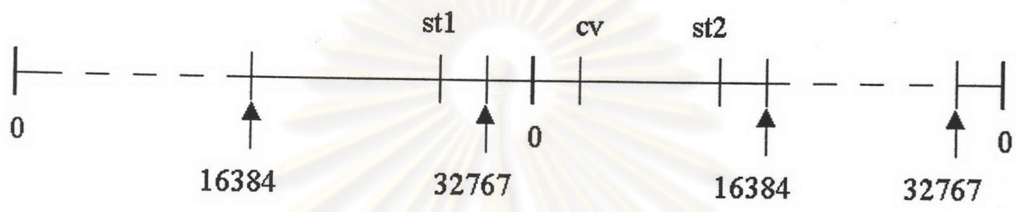
```
outportb(0x43, 0xE2);
st2 = inportb(0x40);
enable();
if ((st1 ^ st2) & 0x80)
      if (cv < 0x4000)
            st1 = st1^0x80;
```

If the variable st1 and st2 occur at the same period, then the output bit in st1 and st2 will be the same. This output bit will also specify the low counting or the high counting as given in Table A.5.

If the variable st1 and st2 occur at the different periods, then the output bits in st1 and st2 may be different and the checking of cv is needed. If the value of cv is less than 16384 (0x4000), the output bit of st1 is the same as that of st2. If the value of cv is greater than or equal to 16384, the output bit of st1 is then used.



(a)



(b)

Fig. A.11 st1 and st2 occur at different period count

Table A.5 The Status Bit D7 in Mode 3 counting

D7 (OUTPUT)	Counting number
0	32768 - 65535
1	0 – 32767

After checking and adjusting the output bit of st1 and st2, the output bit will be checked to the low counting or the high counting with the command below.

```
if ((st1 & 0x80) == 0)/* If on second countdown... */
     cv = cv | 0x8000; /* Set b15 */
```

Finally, the value of cv is returned to the calling program.

```
return cv;
```

<u>Summary of the code for reading the count value</u>

```
unsigned short read_timer0_mode3(void)
{
unsigned char st1, st2; /* Status read-back values */
unsigned short cv;      /* Count value */
disable();              /* No ints please - can use asm cli */
outportb(0x43, 0xE2);   /* Latch and read back status byte */
st1 = inportb(0x40);    /* Read status byte */
outportb(0x43, 0x00);   /* Latch count for timer 0 */
cv = inportb(0x40);     /* Lobyte of count */
cv += inportb(0x40) << 8;      /* Hibyte of count */
cv = ((unsigned short) (0-cv)) >> 1; /*Convert to up-count,0-32767*/
outportb(0x43, 0xE2);   /* Latch and read back status byte */
st2 = inportb(0x40);    /* Read status byte */
enable();               /* Ints back on - can use asm sti */
if ((st1 ^ st2) & 0x80) /* If output pin changed state... */
      if (cv < 0x4000)  /* If reload just occurred... */
            st1 = st1^0x80; /* Use newer output pin status */
if ((st1 & 0x80) == 0)  /* If on second countdown... */
      cv = cv | 0x8000; /* Set b15 */
return cv;              /* Return as up-counter */
}
```

<u>Code Implementation</u>

The code as given above can be applied with the analog to digital conversion (ADC) card. And for the experiment with the fast changing signals, one needs to capture and record a lot of information at a very fine time resolution. For this purpose, a part of C language code can be developed as followed:

```
X = (short *)malloc(NumberOfSamplings*sizeof(short));
/* check X whether or not Null pointer */
T = (unsigned long *)malloc(NumberOfSamplings*sizeof(unsigned long));
```

```
/* check T whether or not Null pointer */
Ti = (float *)malloc(NumberOfSamplings*sizeof(float));
/* check Ti whether or not Null pointer */

/* check trigger signal */

/* loop */
*(X+i) = inport(Address); /* Address specify by the ADC card */
*(T+i) = read_timer0_mode3(void);
/* end loop */

/* if the counting value is repeat 0-65535 in *(T+i)
   then the sum up is needed to receive the value > 65535 */

/* Adjust the count value into the time in microseconds */
/* loop */
*(Ti+i) = *(T+i)*0.8381;

/* write the result into file and free memory */

free(X);free(T);
```

References

[A.1] D.J. Delorie, **DJGPP**, http://www.delorie.com

[A.2] Intel 8254 Programmable Interval Timer,
http://support.intel.com/support/controllers/peripheral/231164.htm

[A.3] Kris Heidenstrom, My PC Timing FAQ (Zip file, release 3),
http://home.clear.net.nz/pages/kheidens

APPENDIX B

MODIFIED TEXAS CODES

B.1 Equation of State for the Thermodynamic Properties of Nitrogen

Span et al. [B.1] developed a formulation for the thermodynamic properties of nitrogen. Many new data sets were available, including high accuracy data from single and dual-sinker apparatuses, which improved the accuracy of the representation of the pρT surface of gaseous, liquid, and supercritical nitrogen, including the saturation states. The speed of sound was measured by spherical resonators, which yielded more accurate information.

The Helmholtz energy (a) had been used to formulate a fundamental property with independent variables of density and temperature. The first equation of state in dimensional form is given by

$$a(\rho,T) = a^0(\rho,T)+a^r(\rho,T) \tag{B.1}$$

where $a^0(\rho,T)$ is the ideal gas contribution to the Helmholtz energy,

and $a^r(\rho,T)$ is the residual Helmholtz energy which corresponds to the influence of intermolecular forces.

All thermodynamic properties can be calculated as derivatives of the Helmholtz energy. The general form of the Helmholtz energy equation is needed to simplify for all property equations. The dimensionless Helmholtz energy (α) with the reduced density (δ) and temperature (τ) as independent variables is a solution. It can be written as followed:

$$\frac{a(\rho,T)}{RT} = \alpha(\delta,\tau) = \alpha^0(\delta,\tau)+\alpha^r(\delta,\tau) \qquad \text{(B.2)}$$

where $\delta = \frac{\rho}{\rho_c}$

ρ_c = critical density, 11.1839 mol/dm^3

$\tau = \frac{T_c}{T}$

T_c = critical temperature, 126.192 K

The correlated equation for the ideal gas Helmholtz energy is shown below:

$$\alpha^0(\delta,\tau) = \ln\delta + a_1\ln\tau + a_2 + a_3\tau + a_4\tau^{-1} + a_5\tau^{-2} + a_6\tau^{-3} + a_7\ln\left[1-\exp(-a_8\tau)\right] \qquad \text{(B.3)}$$

where $a_1 = 2.5$,

$a_2 = -12.769\,527\,08$,

$a_3 = -0.007\,841\,63$,

$a_4 = -1.934\,819\times10^{-4}$,

$a_5 = -1.247\,742\times10^{-5}$,

$a_6 = 6.678\,326\times10^{-8}$,

$a_7 = 1.012\,941$,

and $a_8 = 26.657\,88$

And the correlated equation for the residual Helmholtz energy,

$$\alpha^r(\delta,\tau) = \sum_{k=1}^{6} N_k\delta^{i_k}\tau^{j_k} + \sum_{k=7}^{32} N_k\delta^{i_k}\tau^{j_k}\exp\left(-\delta^{l_k}\right) + \sum_{k=33}^{36} N_k\delta^{i_k}\tau^{j_k}\exp\left(-\phi_k(\delta-1)^2-\beta_k(\tau-\gamma_k)^2\right), \qquad \text{(B.4)}$$

where N_k, i_k, j_k, l_k, ϕ_k, β_k, and γ_k are shown in Table B.1

Table B.1 Parameters and coefficients of the equation of state

k	N_k	i_k	j_k	l_k	ϕ_k	β_k	γ_k
1	0.924 803 575 275	1.0	0.25	0	-	-	-
2	- 0.492 448 489 428	1.0	0.875	0	-	-	-
3	0.661 883 336 938	2.0	0.5	0	-	-	-
4	- 0.192 902 649 201$\times10^{1}$	2.0	0.875	0	-	-	-
5	- 0.622 469 309 629$\times10^{-1}$	3.0	0.375	0	-	-	-
6	0.349 943 957 581	3.0	0.75	0	-	-	-
7	0.564 857 472 498	1.0	0.5	1	-	-	-
8	- 0.161 720 005 987$\times10^{1}$	1.0	0.75	1	-	-	-
9	- 0.481 395 031 883	1.0	2.0	1	-	-	-
10	0.421 150 636 384	3.0	1.25	1	-	-	-
11	- 0.161 962 230 825$\times10^{-1}$	3.0	3.5	1	-	-	-
12	0.172 100 994 165	4.0	1.0	1	-	-	-
13	0.735 448 924 933$\times10^{-2}$	6.0	0.5	1	-	-	-
14	0.168 077 305 479$\times10^{-1}$	6.0	3.0	1	-	-	-
15	- 0.107 626 664 179$\times10^{-2}$	7.0	0.0	1	-	-	-
16	- 0.137 318 088 513$\times10^{-1}$	7.0	2.75	1	-	-	-
17	0.635 466 899 859$\times10^{-3}$	8.0	0.75	1	-	-	-
18	0.304 432 279 419$\times10^{-2}$	8.0	2.5	1	-	-	-
19	- 0.435 762 336 045$\times10^{-1}$	1.0	4.0	2	-	-	-
20	- 0.723 174 889 316$\times10^{-1}$	2.0	6.0	2	-	-	-
21	0.389 644 315 272$\times10^{-1}$	3.0	6.0	2	-	-	-
22	- 0.212 201 363 910$\times10^{-1}$	4.0	3.0	2	-	-	-
23	0.408 822 981 509$\times10^{-2}$	5.0	3.0	2	-	-	-
24	- 0.551 990 017 984$\times10^{-4}$	8.0	6.0	2	-	-	-
25	- 0.462 016 716 479$\times10^{-1}$	4.0	16.0	3	-	-	-
26	- 0.300 311 716 011$\times10^{-2}$	5.0	11.0	3	-	-	-
27	0.368 825 891 208$\times10^{-1}$	5.0	15.0	3	-	-	-
28	- 0.255 856 846 220$\times10^{-2}$	8.0	12.0	3	-	-	-
29	0.896 915 264 558$\times10^{-2}$	3.0	12.0	4	-	-	-
30	- 0.441 513 370 350$\times10^{-2}$	5.0	7.0	4	-	-	-
31	0.133 722 924 858$\times10^{-2}$	6.0	4.0	4	-	-	-
32	0.264 832 491 957$\times10^{-3}$	9.0	16.0	4	-	-	-
33	0.196 688 194 015$\times10^{2}$	1.0	0.0	2	20	325	1.16
34	- 0.209 115 600 730$\times10^{2}$	1.0	1.0	2	20	325	1.16
35	0.167 788 306 989$\times10^{-1}$	3.0	2.0	2	15	300	1.13
36	0.262 767 566 274$\times10^{4}$	2.0	3.0	2	25	275	1.25

With the dimensionless Helmholtz energy equation, all thermodynamics can be derived. The pressure (p) equation, the internal energy (u) equation, the isochoric specific heat capacity (c_v) equation, the isobaric heat capacity (c_p) equation, and the speed of sound (w) equation can be calculated directly as followed:

$$p = \rho RT\left[1+\delta\left(\frac{\partial\alpha^r}{\partial\delta}\right)_\tau\right] \tag{B.5}$$

$$\frac{u}{RT} = \tau\left[\left(\frac{\partial\alpha^0}{\partial\tau}\right)_\delta+\left(\frac{\partial\alpha^r}{\partial\delta}\right)_\tau\right] \tag{B.6}$$

$$\frac{c_v}{R} = -\tau^2\left[\left(\frac{\partial^2\alpha^0}{\partial\tau^2}\right)_\delta+\left(\frac{\partial^2\alpha^r}{\partial\tau^2}\right)_\delta\right] \tag{B.7}$$

$$\frac{c_p}{R} = \frac{c_v}{R}+\frac{\left[1+\delta\left(\frac{\partial\alpha^r}{\partial\delta}\right)_\tau-\delta\tau\left(\frac{\partial^2\alpha^r}{\partial\delta\partial\tau}\right)\right]^2}{\left[1+2\delta\left(\frac{\partial\alpha^r}{\partial\delta}\right)_\tau+\delta^2\left(\frac{\partial^2\alpha^r}{\partial\tau^2}\right)_\tau\right]} \tag{B.8}$$

$$\frac{w^2M}{RT} = 1+2\delta\left(\frac{\partial\alpha^r}{\partial\delta}\right)_\tau+\delta^2\left(\frac{\partial^2\alpha^r}{\partial\tau^2}\right)_\tau-\frac{\left[1+\delta\left(\frac{\partial\alpha^r}{\partial\delta}\right)_\tau-\delta\tau\left(\frac{\partial^2\alpha^r}{\partial\delta\partial\tau}\right)\right]^2}{\tau^2\left[\left(\frac{\partial^2\alpha^0}{\partial\tau^2}\right)_\delta+\left(\frac{\partial^2\alpha^r}{\partial\tau^2}\right)_\delta\right]} \tag{B.9}$$

where	R	= molar gas constant, 8.314 510 J/(mol K)
	M	= molar Mass, 28.01348 g/mol
	p	= pressure in kPa
	u	= internal energy in J/mol
	c_v	= isochoric heat capacity or specific heat at constant volume in J/(mol K)
	c_p	= isobaric heat capacity or specific heat at constant pressure in J/(mol K)
	w	= speed of sound in m/s

A question occurs from the correlation if one finds that a thermodynamic data is enough to identify the state at saturation and also enough to solve for the other thermodynamic data. Then, ancillary equations are needed to calculate the second data. Three ancillary equations are provided here to solve the saturated vapor pressure equation, the saturated liquid density, and the saturated vapor density. These equation are as followed:

$$\ln\left(\frac{p_\sigma}{p_c}\right) = \frac{T_c}{T}\left[N_1\theta + N_2\theta^{1.5} + N_3\theta^{2.5} + N_4\theta^{5}\right] \tag{B.10}$$

where	p_σ	= saturated vapor pressure
	p_c	= critical pressure, 3.3958 MPa
	T	= saturation temperature
	θ	$= 1 - \dfrac{T}{T_c}$
	N_1	= –6.124 452 84,
	N_2	= 1.263 272 20,
	N_3	= –0.765 910 082,
	N_4	= –1.775 705 64

$$\ln\left(\frac{\rho'}{\rho_c}\right) = \left[N_1\theta^{0.3294} + N_2\theta^{2/3} + N_3\theta^{8/3} + N_4\theta^{35/6}\right] \tag{B.11}$$

where	ρ'	= saturated liquid density
	N_1	= 1.486 542b 37,
	N_2	= –0.280 476 066,
	N_3	= 0.089 414 308 5,
	N_4	= –0.119 879 866

$$\ln\left(\frac{\rho''}{\rho_c}\right) = \frac{T_c}{T}\left[N_1\theta^{0.34} + N_2\theta^{5/6} + N_3\theta^{7/6} + N_4\theta^{13/6} + N_5\theta^{14/3}\right] \tag{B.12}$$

where ρ'' = saturated vapor density

$N_1 = -1.701\,271\,64,$

$N_2 = -3.704\,026\,49,$

$N_3 = 1.298\,593\,83,$

$N_4 = -0.561\,424\,977,$

$N_5 = -2.685\,053\,81$

The next question is if two independent variables are not density and temperature, such as pressure and internal energy in computational modeling. And the unknown variables are density or temperature. This will reverse the simple data assignment to solve the equations. Newton-Ralphson method can be applied as a tool to answer this question. The concept of the method starts from two following equations:

$$p_{known} \approx p(\delta_0,\tau_0) + \left.\frac{\partial p}{\partial \delta}\right|_{\delta_0} d\delta + \left.\frac{\partial p}{\partial \tau}\right|_{\tau_0} d\tau \tag{B.13}$$

$$u_{known} \approx u(\delta_0,\tau_0) + \left.\frac{\partial u}{\partial \delta}\right|_{\delta_0} d\delta + \left.\frac{\partial u}{\partial \tau}\right|_{\tau_0} d\tau \tag{B.14}$$

And the brief procedure can be summarized:

1. Initiate appropriate δ_0 and τ_0
2. Calculate $p(\delta_0,\tau_0)$, $u(\delta_0,\tau_0)$, $\left.\frac{\partial p}{\partial \delta}\right|_{\delta_0}$, $\left.\frac{\partial p}{\partial \tau}\right|_{\tau_0}$, $\left.\frac{\partial u}{\partial \delta}\right|_{\delta_0}$, and $\left.\frac{\partial u}{\partial \tau}\right|_{\tau_0}$
3. Check $|p_{known} - p(\delta_0,\tau_0)|$ and $|u_{known} - u(\delta_0,\tau_0)|$ with a tolerance
4. Find $d\delta$ and $d\tau$ from $\begin{bmatrix} p_{known} - p(\delta_0,\tau_0) \\ u_{known} - u(\delta_0,\tau_0) \end{bmatrix} = \begin{bmatrix} \left.\frac{\partial p}{\partial \delta}\right|_{\delta_0} & \left.\frac{\partial p}{\partial \tau}\right|_{\tau_0} \\ \left.\frac{\partial u}{\partial \delta}\right|_{\delta_0} & \left.\frac{\partial u}{\partial \tau}\right|_{\tau_0} \end{bmatrix} \begin{bmatrix} d\delta \\ d\tau \end{bmatrix}$
5. Update new δ and new τ from $\delta_{new} = \delta_0 + d\delta$ and $\tau_{new} = \tau_0 + d\tau$
6. Repeat step 2. and 3. with δ_{new} and τ_{new} until $|p_{known} - p(\delta_{new},\tau_{new})|$ and $|u_{known} - u(\delta_{new},\tau_{new})|$ less than the tolerance.

B.2 Modified Subroutines in TEXAS code

```
 function gas_internalenergysat(idummy)
   include 'fcicom.for'
   !SET INTERNAL ENERGY OF GAS PHASE AT PSAT, p(idummy)
   !finding gas saturated temperature,tsatN2 from p(idummy)
   psat0N2      = p(idummy)/1.0D6
if (psat0N2.LE.N2PC) then
   tsatN2       = N1Psat*N2TC/(log(psat0N2/N2PC)+N1Psat)      !first approximation
   thetaN2      = 1.0-tsatN2/N2TC                            !first approximation
   if (thetaN2.LT.0.0 ) then
     tsatN2     = N2TC
     thetaN2    = 0.0 !1.0-tsatN2/N2TC
   endif

   psatN2       = N2PC*exp((N2TC/tsatN2)*(N1Psat*thetaN2 &
                  +N2Psat*(thetaN2**1.5)+N3Psat*(thetaN2**2.5) &
                  +N4Psat*(thetaN2**5)))

   do while (dabs(psatN2 - psat0N2).GT.0.000001) !1 Pascal tolerance
         dpsbydtN2=(-1.0)*(N2TC/(tsatN2**2.0))* &
                    (N1Psat*thetaN2     +N2Psat*thetaN2**1.5 &
                    +N3Psat*thetaN2**2.5+N4Psat*thetaN2**5.0) &
                    -(1.0/tsatN2)*(N1Psat+N2Psat*1.5*thetaN2**0.5 &
                                 +N3Psat*2.5*thetaN2**1.5+N4Psat*5.0*thetaN2**4.0)
        dpsbydtN2=psatN2*dpsbydtN2

        tsatN2    = (psat0N2-psatN2)/dpsbydtN2+tsatN2
        thetaN2   = 1-tsatN2/N2TC

        psatN2    = N2PC*exp((N2TC/tsatN2)*(N1Psat*thetaN2 &
                    +N2Psat*(thetaN2**1.5)+N3Psat*(thetaN2**2.5) &
                    +N4Psat*(thetaN2**5)))
   end do

   !finding saturated vapor density (mol/dm3) from tsatN2
   thetaN2    = 1-tsatN2/N2TC
   rhogsatN2 = N2RHOC*exp((N2TC/tsatN2) &
              *(N1Vsat*(thetaN2**0.34) &
                +N2Vsat*(thetaN2**(5.0/6.0)) &
                +N3Vsat*(thetaN2**(7.0/6.0)) &
                +N4Vsat*(thetaN2**(13.0/6.0)) &
                +N5Vsat*(thetaN2**(14.0/3.0))))
   deltaN2 = rhogsatN2/N2RHOC
   torN2    = N2TC/tsatN2
   call N2assign
   ugsatN2 = R*tsatN2*(torda0bydt()+tordarbydt())             !in J/mol
   siests(idummy)          = ugsatN2*1000.0/N2MolarMass+ushift !in J/kg
   gas_internalenergysat = siests(idummy)                     !in J/kg
else
   ugsatN2                 = 515.3                            !in J/mol
   siests(idummy)          = 18393.8                          !in J/kg
   siests(idummy)          = 18393.8+ushift
   gas_internalenergysat = siests(idummy)                     !in J/kg
endif

end function gas_internalenergysat

function gas_internalenergy(idummy)
  include 'fcicom.for'
  !SET GAS PHASE INTERNAL ENERGY from p(idummy) and tg(idummy)
  pN2       = p(idummy)/1.0D3
  tN2       = tg(idummy)

  !finding saturated temperature under critical pressure condition
  if (p(idummy)/1.0D6.LE.N2PC) then
    psat0N2 = p(idummy)/1.0D6
    tsatN2  = N1Psat*N2TC/(log(psat0N2/N2PC)+N1Psat) !first approximation
    thetaN2 = 1.0-tsatN2/N2TC                         !first approximation
   if (thetaN2.LT.0.0 ) then
     tsatN2    = N2TC
     thetaN2   = 0.0 !1.0-tsatN2/N2TC
   endif

   psatN2      = N2PC*exp((N2TC/tsatN2)*(N1Psat*thetaN2 &
                 +N2Psat*(thetaN2**1.5)+N3Psat*(thetaN2**2.5) &
                 +N4Psat*(thetaN2**5)))

   do while (dabs(psatN2 - psat0N2).GT.0.000001) !1 Pascal tolerance
         dpsbydtN2=(-1.0)*(N2TC/(tsatN2**2.0))* &
                  (N1Psat*thetaN2      +N2Psat*thetaN2**1.5 &
                   +N3Psat*thetaN2**2.5+N4Psat*thetaN2**5.0) &
                   -(1.0/tsatN2)*(N1Psat+N2Psat*1.5*thetaN2**0.5 &
                  +N3Psat*2.5*thetaN2**1.5+N4Psat*5.0*thetaN2**4.0)
         dpsbydtN2=psatN2*dpsbydtN2

         tsatN2    = (psat0N2-psatN2)/dpsbydtN2+tsatN2
         thetaN2   = 1-tsatN2/N2TC
     if (thetaN2.LT.0.0 ) then
       tsatN2  = N2TC
       thetaN2 = 0.0 !1.0-tsatN2/N2TC
     endif
        psatN2 = N2PC*exp((N2TC/tsatN2)*(N1Psat*thetaN2 &
                        +N2Psat*(thetaN2**1.5)+N3Psat*(thetaN2**2.5) &
                        +N4Psat*(thetaN2**5)))
   end do
 else
   tsatN2 = N2TC
 endif

 !finding gas_internalenergy
 if (p(idummy)/1.0D6.LE.N2PC.AND.tsatN2.GT.tN2) then
 !specific internal energy at saturation will be assigned.
      rhogsatN2 = N2RHOC*exp((N2TC/tsatN2) &
                            *(N1Vsat*(thetaN2**0.34) &
                             +N2Vsat*(thetaN2**(5.0/6.0)) &
                             +N3Vsat*(thetaN2**(7.0/6.0)) &
                             +N4Vsat*(thetaN2**(13.0/6.0)) &
                             +N5Vsat*(thetaN2**(14.0/3.0))))
      deltaN2   = rhogsatN2/N2RHOC
      torN2     = N2TC/tsatN2
      call N2assign
      usatN2 = R*tsatN2*(torda0bydt()+tordarbydt())  !in J/mol
   gas_internalenergy = usatN2*1000.0/N2MolarMass + ushift !in J/kg
      rhogN2 = rhogsatN2
      uN2    = usatN2
else !superheat vapor or above critical conditions
      if (p(idummy)/1.0D6.GT.N2PC.AND.tN2.LE.N2TC) tN2 = N2TC
      rhogN2 = 1.0
      deltaN2  = rhogN2/N2RHOC
      torN2    = N2TC/tN2
      call N2assign
      var0N2     = pN2/(N2RHOC*R*tN2)
      varN2      = deltaN2*(1.0+deltadarbydd())
      do while (dabs(var0N2-varN2).GT.0.0000001)
              dvarN2bydd = 1+2.0*deltadarbydd()+delta2d2arbydd2()
              ddeltaN2    = (var0N2-varN2)/dvarN2bydd
              deltaN2     = deltaN2+ddeltaN2
              rhogN2     = N2RHOC*deltaN2
              if (rhogN2.LT.0.01) rhogN2 = 0.01
              if (rhogN2.GT.42.0) rhogN2 = 42.0
              deltaN2     = rhogN2/N2RHOC
              call N2assign
              varN2       = deltaN2*(1.0+deltadarbydd())
      end do
      rhogN2 = deltaN2*N2RHOC

      uN2 = R*tN2*(torda0bydt()+tordarbydt()) !in J/mol
      gas_internalenergy = uN2*1000.0/N2MolarMass + ushift !in J/kg
      if (gas_internalenergy.LT.0.0) gas_internalenergy = 0.0
endif
```

```
end function gas_internalenergy

function gas_internalenergyreset(idummy)
  include 'fcicom.for'
  !reset int. energy to sat. if u is below u_sat
  !??? this doesn't make sense anymore since we have ncgs
  if (sieg(idummy).lt.siegs(idummy)) then
     gas_internalenergyreset=siegs(idummy)
     print*, 'I am here in gas_internalenergyreset'
  else
     gas_internalenergyreset=sieg(idummy)
  end if
end function gas_internalenergyreset

!function for saturation nitrogen
function gas_tempsat(idummy)
  include 'fcicom.for'
  !calculate the saturation temperature of nitrogen
  psat0N2            = p(idummy)/1.0D6
  if (psat0N2.LE.N2PC) then
    gas_tempsat = N1Psat*N2TC/(log(psat0N2/N2PC)+N1Psat)!first approximation
    thetaN2     = 1.0-gas_tempsat/N2TC                             !first
approximation
    if (thetaN2.LT.0.0 ) then
      gas_tempsat = N2TC
      thetaN2     = 0.0 !1.0-gas_tempsat/N2TC
    endif

    psatN2      = N2PC*exp((N2TC/gas_tempsat)*(N1Psat*thetaN2 &
                       +N2Psat*(thetaN2**1.5)+N3Psat*(thetaN2**2.5) &
                       +N4Psat*(thetaN2**5)))

    do while (dabs(psatN2 - psat0N2).GT.0.000001) !1 Pascal tolerance
         dpsbydtN2=(-1.0)*(N2TC/(gas_tempsat**2.0))* &
                   (N1Psat*thetaN2      +N2Psat*thetaN2**1.5 &
                          +N3Psat*thetaN2**2.5+N4Psat*thetaN2**5.0) &
                  -(1.0/gas_tempsat)*(N1Psat+N2Psat*1.5*thetaN2**0.5 &
                          +N3Psat*2.5*thetaN2**1.5+N4Psat*5.0*thetaN2**4.0)
         dpsbydtN2=psatN2*dpsbydtN2

         gas_tempsat  = (psat0N2-psatN2)/dpsbydtN2+gas_tempsat
         thetaN2      = 1-gas_tempsat/N2TC

         psatN2 = N2PC*exp((N2TC/gas_tempsat)*(N1Psat*thetaN2 &
                       +N2Psat*(thetaN2**1.5)+N3Psat*(thetaN2**2.5) &
                       +N4Psat*(thetaN2**5)))
    end do
  else
    gas_tempsat = N2TC
  endif
end function gas_tempsat

function gas_temp(idummy)
  include 'fcicom.for'
  pN2   = p(idummy)/1000.0                              !pN2 in kPa
  uN2   = (sieg(idummy)-ushift)*N2MolarMass/1000.0      !uN2 in J/mol

!calculate ugsatN2
  psat0N2          = p(idummy)/1.0D6
if (psat0N2.LE.N2PC) then
  tsatN2       = N1Psat*N2TC/(log(psat0N2/N2PC)+N1Psat) !first approximation
  thetaN2      = 1.0-tsatN2/N2TC                        !first approximation
  if (thetaN2.LT.0.0 ) then
    tsatN2     = N2TC
    thetaN2    = 0.0 !1.0-tsatN2/N2TC
  endif

  psatN2         = N2PC*exp((N2TC/tsatN2)*(N1Psat*thetaN2 &
                       +N2Psat*(thetaN2**1.5)+N3Psat*(thetaN2**2.5) &
                       +N4Psat*(thetaN2**5)))

  do while (dabs(psatN2 - psat0N2).GT.0.000001) !1 Pascal tolerance
       dpsbydtN2=(-1.0)*(N2TC/(tsatN2**2.0))* &
```

```
                   (N1Psat*thetaN2       +N2Psat*thetaN2**1.5 &
                   +N3Psat*thetaN2**2.5 +N4Psat*thetaN2**5.0) &
                   -(1.0/tsatN2)*(N1Psat+N2Psat*1.5*thetaN2**0.5 &
                   +N3Psat*2.5*thetaN2**1.5+N4Psat*5.0*thetaN2**4.0)
        dpsbydtN2=psatN2*dpsbydtN2

        tsatN2  = (psat0N2-psatN2)/dpsbydtN2+tsatN2
        thetaN2 = 1-tsatN2/N2TC

        psatN2  = N2PC*exp((N2TC/tsatN2)*(N1Psat*thetaN2 &
                           +N2Psat*(thetaN2**1.5)+N3Psat*(thetaN2**2.5) &
                           +N4Psat*(thetaN2**5)))
  end do

  !finding saturated vapor density (mol/dm3) from tsatN2
  thetaN2   = 1-tsatN2/N2TC
  rhogsatN2 = N2RHOC*exp((N2TC/tsatN2) &
                         *(N1Vsat*(thetaN2**0.34) &
                         +N2Vsat*(thetaN2**(5.0/6.0)) &
                         +N3Vsat*(thetaN2**(7.0/6.0)) &
                         +N4Vsat*(thetaN2**(13.0/6.0)) &
                         +N5Vsat*(thetaN2**(14.0/3.0))))
  deltaN2 = rhogsatN2/N2RHOC
  torN2   = N2TC/tsatN2
  call N2assign
  ugsatN2 = R*tsatN2*(torda0bydt()+tordarbydt())                  !in J/mol
  ugsatN2 = ugsatN2*1000.0/N2MolarMass                            !in J/kg
else
  ugsatN2 = 515.3                                                 !in J/mol
  ugsatN2 = 18393.8                                               !in J/kg
endif

  if (sieg(idummy) .LE. ugsatN2) then
    gas_temp = tsatN2
  else
       if (sieg(idummy) < 0.0) then !liquid phase
              gas_temp       = 63.170
              rhogN2         = 30.960
       else
              gas_temp       = 100.0
              rhogN2         = 0.12268
       endif
400    torN2          = N2TC/gas_temp
       deltaN2      = rhogN2/N2RHOC

300    call N2assign

       p_newN2 = rhogN2*R*gas_temp*(1.0+deltadarbydd())
       u_newN2 = R*gas_temp*(torda0bydt()+tordarbydt())

       dpN2    = pN2 - p_newN2
       duN2    = uN2 - u_newN2
       devdpN2 = dabs(dpN2/pN2)
       devduN2 = dabs(duN2/uN2)

       if (devduN2.LT.0.00001.AND.devdpN2.LT.0.00001) go to 200

       dpbydtN2  = (rhogN2*R*N2TC/torN2**2.0)* &
                   deltatord2arbyddt() - p_newN2/torN2
       dpbyddN2  = N2RHOC*R*gas_temp*(1.0 + 2.0*deltadarbydd() + &
                   delta2d2arbydd2())

       dubydtN2  = (R*N2TC/torN2**2.0)* &
                   (tor2d2a0bydt2()+tor2d2arbydt2())
       dubyddN2  = (R*N2TC/(torN2*deltaN2))* &
                   deltatord2arbyddt()

       dtorN2    = (-dpN2*dubyddN2+duN2*dpbyddN2) &
                   /(dubydtN2*dpbyddN2-dpbydtN2*dubyddN2)
       ddeltaN2  = ( dpN2*dubydtN2-duN2*dpbydtN2) &
                   /(dubydtN2*dpbyddN2-dpbydtN2*dubyddN2)

       torN2    = torN2    + dtorN2  !    tor = TC/T
```

```
        deltaN2 = deltaN2 + ddeltaN2

        if (torN2.LT.N2TC/1200.0 .AND. deltaN2.LT.0.01/N2RHOC) then
!               write (*,*) '1'
                gas_temp   = 1200.0
                rhogN2 = 0.01
                go to 400
        endif

        if (torN2.LT.N2TC/1200.0 .AND. deltaN2.GT.45.0/N2RHOC) then
!               write (*,*) '2'
                gas_temp   = 1200.0
                rhogN2 = 45.0
                go to 400
        endif

        if (torN2.GT.N2TC/60.0 .AND. deltaN2.LT.0.01/N2RHOC) then
!               write (*,*) '3'
                gas_temp   = 60.0
                rhogN2 = 0.01
                go to 400
        endif

        if (torN2.GT.N2TC/60.0 .AND. deltaN2.GT.42.0/N2RHOC) then
!               write (*,*) '4'
                gas_temp   = 60.0
                rhogN2 = 45.0
                go to 400
        endif

        if (torN2.LT.N2TC/1200.0) then
!               write (*,*) '5'
                gas_temp   = 1200.0
                rhogN2 = N2RHOC*deltaN2
                go to 400
        endif

        if (torN2.GT.N2TC/60.0) then
!               write (*,*) '6'
                gas_temp   = 60.0
                rhogN2 = N2RHOC*deltaN2
                go to 400
        endif

        if (deltaN2.LT.0.01/N2RHOC) then
!               write (*,*) '7'
                gas_temp   = N2TC/torN2
                rhogN2 = 0.01
                go to 400
        endif

        if (deltaN2.GT.45.0/N2RHOC) then
!               write (*,*) '8'
                gas_temp   = N2TC/torN2
                rhogN2 = 45.0
                go to 400
        endif

        gas_temp  = N2TC/torN2
        rhogN2    = deltaN2*N2RHOC

        go to 300

200 gas_temp   = N2TC/torN2
        rhogN2  = deltaN2*N2RHOC
        endif

end function gas_temp

function gas_density(idummy)
  include 'fcicom.for'
  pN2       = p(idummy)/1.0D3
  tN2       = tg(idummy)
  !assigning initial density within liquid, vapor, and critical region
  if     (tN2.GE.N2TC  .AND.  pN2.GE.N2PC*1000.0) then !above critical
       rhogN2 = N2RHOC !uses critical state for initial
       lN2    = 2
  elseif (tN2.GE.N2TC  .AND.  pN2.LT.N2PC*1000.0) then !vapor phase
       rhogN2 = 0.01   !or uses 44.0 but 0.01 is nearer
       lN2    = 1
  elseif (tN2.LT.N2TC  .AND.  pN2.GT.N2PC*1000.0) then !liquid phase
       rhogN2 = 44.0   !uses liquid phase for initial
       lN2    = 1
  else
       thetaN2 = 1-tN2/N2TC
       psatN2  = N2PC*exp((N2TC/tN2) &
                      *(N1Psat*thetaN2 &
                        +N2Psat*(thetaN2**1.5) &
                        +N3Psat*(thetaN2**2.5) &
                        +N4Psat*(thetaN2**5)))
       if (pN2 .GT. psatN2*1000.0) then
         rhogN2 = 44.0 !initial liquid phase
         lN2    = 0
         !The highest bound of density observed from N2 table produced by
         !SPAN et al. is 44.0 mol/dm3 in gaseous phase.
       else
         rhogN2 = 0.01 !initial gaseous phase
         lN2    = 1
         !The lowest bound of density observed from N2 table produced by
         !SPAN et al. is 0.01 mol/dm3 in gaseous phase.
       endif
  endif

!In TEXAS, the rg(i) depends on p(i)/R*tg(i) only, even though
!the state is in liquid. Anyhow, this modified module uses the
!equations from Roland Span in the gaseous phase properties in
!the gaseous phase and bounds the gas density by saturation pressure
!inside the liquid phase region.
  if (lN2.EQ.1.OR.lN2.EQ.2) then
       deltaN2  = rhogN2/N2RHOC
       torN2    = N2TC/tN2
       call N2assign
       var0N2   = pN2/(N2RHOC*R*tN2)
       varN2    = deltaN2*(1.0+deltadarbydd())

       do while (dabs(var0N2-varN2).GT.0.0000001)
         dvarN2bydd  = 1+2.0*deltadarbydd()+delta2d2arbydd2()
         ddeltaN2    = (var0N2-varN2)/dvarN2bydd
         deltaN2     = deltaN2+ddeltaN2
         rhogN2        = N2RHOC*deltaN2
         if (rhogN2.LT.0.01) rhogN2 = 0.01
         if (rhogN2.GT.42.0) rhogN2 = 42.0
         deltaN2     = rhogN2/N2RHOC
         call N2assign
         varN2       = deltaN2*(1.0+deltadarbydd())
       end do

       rhogN2        = deltaN2*N2RHOC             !in mol/dm3
       gas_density   = rhogN2*N2MolarMass   !in g/dm3 = kg/m3

  elseif (lN2.EQ.0) then
       psat0N2       = p(idummy)/1.0D6
       tsatN2        = N1Psat*N2TC/(log(psat0N2/N2PC)+N1Psat)
       !tsatN2 first approximation
       thetaN2     = 1.0-tsatN2/N2TC
       !first  approximation
       if (thetaN2.LT.0.0 ) then
     tsatN2    = N2TC
         thetaN2       = 0.0 !1.0-tsatN2/N2TC
    endif

    psatN2      = N2PC*exp((N2TC/tsatN2)*(N1Psat*thetaN2 &
                          +N2Psat*(thetaN2**1.5)+N3Psat*(thetaN2**2.5) &
                          +N4Psat*(thetaN2**5)))
```

```
        do while (dabs(psatN2 - psat0N2).GT.0.000001) !1 Pascal tolerance
          dpsbydtN2=(-1.0)*(N2TC/(tsatN2**2.0))* &
                   (N1Psat*thetaN2     +N2Psat*thetaN2**1.5 &
                   +N3Psat*thetaN2**2.5+N4Psat*thetaN2**5.0) &
                   -(1.0/tsatN2)*(N1Psat+N2Psat*1.5*thetaN2**0.5 &
                   +N3Psat*2.5*thetaN2**1.5+N4Psat*5.0*thetaN2**4.0)
          dpsbydtN2=psatN2*dpsbydtN2

          tsatN2        = (psat0N2-psatN2)/dpsbydtN2+tsatN2
          thetaN2       = 1-tsatN2/N2TC

          psatN2 = N2PC*exp((N2TC/tsatN2)*(N1Psat*thetaN2 &
                         +N2Psat*(thetaN2**1.5)+N3Psat*(thetaN2**2.5) &
                         +N4Psat*(thetaN2**5)))
        end do
        !finding saturated vapor density at p(idummy)
        !the TEXAS should not use the gas_density function
        !if the state is liquid.
        rhogN2 = N2RHOC*exp((N2TC/tsatN2) &
                         *(N1Vsat*(thetaN2**0.34) &
                          +N2Vsat*(thetaN2**(5.0/6.0)) &
                          +N3Vsat*(thetaN2**(7.0/6.0)) &
                          +N4Vsat*(thetaN2**(13.0/6.0)) &
                          +N5Vsat*(thetaN2**(14.0/3.0))))
        gas_density = rhogN2*N2MolarMass      !in g/dm3 = kg/m3
  endif

end function gas_density

function gas_specheat(idummy)
  include 'fcicom.for'
  pN2      = p(idummy)/1.0D3
  tN2      = tg(idummy)

  !assigning initial density within liquid, vapor, and critical region
  if     (tN2.GE.N2TC  .AND.  pN2.GE.N2PC*1000.0) then !above critical
       rhogN2 = N2RHOC !uses critical state for initial
       lN2    = 2
  elseif (tN2.GE.N2TC  .AND.  pN2.LT.N2PC*1000.0) then !vapor phase
       rhogN2 = 0.01   !or uses 44.0 but 0.01 is nearer
       lN2    = 1
  elseif (tN2.LT.N2TC  .AND.  pN2.GT.N2PC*1000.0) then !liquid phase
       rhogN2 = 44.0   !uses liquid phase for initial
       lN2    = 1
  else
       thetaN2 = 1-tN2/N2TC
       psatN2  = N2PC*exp((N2TC/tN2) &
                      *(N1Psat*thetaN2 &
                       +N2Psat*(thetaN2**1.5) &
                       +N3Psat*(thetaN2**2.5) &
                       +N4Psat*(thetaN2**5)))
       if (pN2 .GT. psatN2*1000.0) then
         rhogN2 = 44.0 !initial liquid phase
         lN2    = 0
         !The highest bound of density observed from N2 table produced by
         !SPAN et al. is 44.0 mol/dm3 in gaseous phase.
       else
         rhogN2 = 0.01 !initial gaseous phase
         lN2    = 1
         !The lowest bound of density observed from N2 table produced by
         !SPAN et al. is 0.01 mol/dm3 in gaseous phase.
       endif
  endif

!In TEXAS, the rg(i) depends on p(i)/R*tg(i) only, even though
!the state is in liquid. Anyhow, this modified module uses the
!equations from Roland Span in the gaseous phase properties in
!the gaseous phase and bounds the gas density by saturation pressure
!inside the liquid phase region.
  if (lN2.EQ.1.OR.lN2.EQ.2) then
       deltaN2  = rhogN2/N2RHOC
       torN2    = N2TC/tN2
       call N2assign
```

```
      var0N2   = pN2/(N2RHOC*R*tN2)
      varN2    = deltaN2*(1.0+deltadarbydd())

      do while (dabs(var0N2-varN2).GT.0.0000001)
        dvarN2bydd  = 1+2.0*deltadarbydd()+delta2d2arbydd2()
        ddeltaN2    = (var0N2-varN2)/dvarN2bydd
        deltaN2     = deltaN2+ddeltaN2
        rhogN2      = N2RHOC*deltaN2
        if (rhogN2.LT.0.01) rhogN2 = 0.01
        if (rhogN2.GT.42.0) rhogN2 = 42.0
        deltaN2     = rhogN2/N2RHOC
        call N2assign
        varN2       = deltaN2*(1.0+deltadarbydd())
      end do

      rhogN2        = deltaN2*N2RHOC          !in mol/dm3
  elseif (lN2.EQ.0) then
      psat0N2       = p(idummy)/1.0D6
      tsatN2        = N1Psat*N2TC/(log(psat0N2/N2PC)+N1Psat)
      !tsatN2 first approximation
      thetaN2      = 1.0-tsatN2/N2TC
      !first  approximation
      if (thetaN2.LT.0.0 ) then
        tsatN2     = N2TC
        thetaN2    = 0.0 !1.0-tsatN2/N2TC
    endif

    psatN2      = N2PC*exp((N2TC/tsatN2)*(N1Psat*thetaN2 &
                       +N2Psat*(thetaN2**1.5)+N3Psat*(thetaN2**2.5) &
                       +N4Psat*(thetaN2**5)))

      do while (dabs(psatN2 - psat0N2).GT.0.000001) !1 Pascal tolerance
        dpsbydtN2=(-1.0)*(N2TC/(tsatN2**2.0))* &
                 (N1Psat*thetaN2      +N2Psat*thetaN2**1.5 &
                 +N3Psat*thetaN2**2.5+N4Psat*thetaN2**5.0) &
                 -(1.0/tsatN2)*(N1Psat+N2Psat*1.5*thetaN2**0.5 &
                        +N3Psat*2.5*thetaN2**1.5+N4Psat*5.0*thetaN2**4.0)
        dpsbydtN2=psatN2*dpsbydtN2

        tsatN2        = (psat0N2-psatN2)/dpsbydtN2+tsatN2
        thetaN2       = 1-tsatN2/N2TC

        psatN2 = N2PC*exp((N2TC/tsatN2)*(N1Psat*thetaN2 &
                          +N2Psat*(thetaN2**1.5)+N3Psat*(thetaN2**2.5) &
                          +N4Psat*(thetaN2**5)))
      end do
      !finding saturated vapor density at p(idummy)
      !the TEXAS should not use the gas_density function
      !if the state is liquid.
      rhogN2 = N2RHOC*exp((N2TC/tsatN2) &
                       *(N1Vsat*(thetaN2**0.34) &
                         +N2Vsat*(thetaN2**(5.0/6.0)) &
                         +N3Vsat*(thetaN2**(7.0/6.0)) &
                         +N4Vsat*(thetaN2**(13.0/6.0)) &
                         +N5Vsat*(thetaN2**(14.0/3.0))))
      tN2 = tsatN2
  endif

  deltaN2  = rhogN2/N2RHOC
  torN2    = N2TC/tN2
  call N2assign
  x1N2 = deltadarbydd()
  x4N2 = tor2d2a0bydt2()
  x5N2 = tor2d2arbydt2()
  x6N2 = deltatord2arbydddt()
  x7N2 = delta2d2arbydd2()
  x8N2 = (1.0+x1N2-x6N2)**2.0
  cpN2 = R*((-1.0)*(x4N2+x5N2) &
       +x8N2/(1.0+2.0*x1N2+x7N2))
  gas_specheat = cpN2/(N2MolarMass*1000.0)
end function gas_specheat
```

```
function gas_recipsoundspeedsq(idummy)
  include 'fcicom.for'
  pN2      = p(idummy)/1.0D3
  tN2      = tg(idummy)

  !assigning initial density within liquid, vapor, and critical region
  if      (tN2.GE.N2TC  .AND.  pN2.GE.N2PC*1000.0) then !above critical
        rhogN2 = N2RHOC !uses critical state for initial
        lN2    = 2
  elseif (tN2.GE.N2TC  .AND.  pN2.LT.N2PC*1000.0) then !vapor phase
        rhogN2 = 0.01    !or uses 44.0 but 0.01 is nearer
        lN2    = 1
  elseif (tN2.LT.N2TC  .AND.  pN2.GT.N2PC*1000.0) then !liquid phase
        rhogN2 = 44.0    !uses liquid phase for initial
        lN2    = 1
  else
        thetaN2 = 1-tN2/N2TC
        psatN2  = N2PC*exp((N2TC/tN2) &
                          *(N1Psat*thetaN2 &
                            +N2Psat*(thetaN2**1.5) &
                            +N3Psat*(thetaN2**2.5) &
                            +N4Psat*(thetaN2**5)))
        if (pN2 .GT. psatN2*1000.0) then
          rhogN2 = 44.0 !initial liquid phase
          lN2    = 0
          !The highest bound of density observed from N2 table produced by
          !SPAN et al. is 44.0 mol/dm3 in gaseous phase.
        else
          rhogN2 = 0.01 !initial gaseous phase
          lN2    = 1
          !The lowest bound of density observed from N2 table produced by
          !SPAN et al. is 0.01 mol/dm3 in gaseous phase.
        endif
  endif

!In TEXAS, the rg(i) depends on p(i)/R*tg(i) only, even though
!the state is in liquid. Anyhow, this modified module uses the
!equations from Roland Span in the gaseous phase properties in
!the gaseous phase and bounds the gas density by saturation pressure
!inside the liquid phase region.
  if (lN2.EQ.1.OR.lN2.EQ.2) then
        deltaN2  = rhogN2/N2RHOC
        torN2    = N2TC/tN2
        call N2assign
        var0N2   = pN2/(N2RHOC*R*tN2)
        varN2    = deltaN2*(1.0+deltadarbydd())

        do while (dabs(var0N2-varN2).GT.0.0000001)
          dvarN2bydd   = 1+2.0*deltadarbydd()+delta2d2arbydd2()
          ddeltaN2     = (var0N2-varN2)/dvarN2bydd
          deltaN2      = deltaN2+ddeltaN2
          rhogN2       = N2RHOC*deltaN2
          if (rhogN2.LT.0.01) rhogN2 = 0.01
          if (rhogN2.GT.42.0) rhogN2 = 42.0
          deltaN2      = rhogN2/N2RHOC
          call N2assign
          varN2        = deltaN2*(1.0+deltadarbydd())
        end do

        rhogN2        = deltaN2*N2RHOC        !in mol/dm3
  elseif (lN2.EQ.0) then
        psat0N2       = p(idummy)/1.0D6
        tsatN2        = N1Psat*N2TC/(log(psat0N2/N2PC)+N1Psat)
        !tsatN2 first approximation
        thetaN2        = 1.0-tsatN2/N2TC
        !first  approximation
        if (thetaN2.LT.0.0 ) then
      tsatN2    = N2TC
          thetaN2       = 0.0 !1.0-tsatN2/N2TC
    endif

    psatN2      = N2PC*exp((N2TC/tsatN2)*(N1Psat*thetaN2 &
                          +N2Psat*(thetaN2**1.5)+N3Psat*(thetaN2**2.5) &
```

```
                  +N4Psat*(thetaN2**5)))

        do while (dabs(psatN2 - psat0N2).GT.0.000001) !1 Pascal tolerance
          dpsbydtN2=(-1.0)*(N2TC/(tsatN2**2.0))* &
                    (N1Psat*thetaN2     +N2Psat*thetaN2**1.5 &
                   +N3Psat*thetaN2**2.5+N4Psat*thetaN2**5.0) &
                    -(1.0/tsatN2)*(N1Psat+N2Psat*1.5*thetaN2**0.5 &
                   +N3Psat*2.5*thetaN2**1.5+N4Psat*5.0*thetaN2**4.0)
          dpsbydtN2=psatN2*dpsbydtN2

          tsatN2       = (psat0N2-psatN2)/dpsbydtN2+tsatN2
          thetaN2      = 1-tsatN2/N2TC

          psatN2 = N2PC*exp((N2TC/tsatN2)*(N1Psat*thetaN2 &
                        +N2Psat*(thetaN2**1.5)+N3Psat*(thetaN2**2.5) &
                        +N4Psat*(thetaN2**5)))
        end do
        !finding saturated vapor density at p(idummy)
        !the TEXAS should not use the gas_density function
        !if the state is liquid.
        rhogN2 = N2RHOC*exp((N2TC/tsatN2) &
                 *(N1Vsat*(thetaN2**0.34) &
                 +N2Vsat*(thetaN2**(5.0/6.0)) &
                 +N3Vsat*(thetaN2**(7.0/6.0)) &
                 +N4Vsat*(thetaN2**(13.0/6.0)) &
                 +N5Vsat*(thetaN2**(14.0/3.0))))
        tN2 = tsatN2
  endif

  deltaN2  = rhogN2/N2RHOC
  torN2    = N2TC/tN2
  call N2assign
  x1N2  = deltadarbydd()
  x4N2  = tor2d2a0bydt2()
  x5N2  = tor2d2arbydt2()
  x6N2  = deltatord2arbyddt()
  x7N2  = delta2d2arbydd2()
  x8N2  = (1.0+x1N2-x6N2)**2.0
  wsqN2 = R*tN2*1000.0*(1.0+2.0*x1N2+x7N2-x8N2/(x4N2+x5N2))/N2MolarMass
  gas_recipsoundspeedsq = 1.0/wsqN2

end function gas_recipsoundspeedsq

subroutine setupmix
  include 'fcicom.for'

  al(v1,v2)=(dx(im)*v1+dx(i)*v2)/dxl
  !set up dx, xb arrays
  k1=1
  dxmin=1e10
  do k=1,ngmax
     if (ndx(k).eq.0) go to 205
     nreg=ndx(k)
     dxg=dxi(k)
     dxmin=amin1(dxg,dxmin)
     if (k.eq.1) dx(1)=dxi(1)
     do kk=1,nreg
        if (kk+k1.gt.nmax) go to 95
        dx(k1+kk)=dxg
     end do
     k1=k1+nreg
  end do
205 continue
  k1=k1+1
  dx(k1)=dxg
  !setup the boundary cell length by specific datas for
  !constant pressure and free gradient boundary condition
  if (flb.ne.2) dx(1)=dx(2)
  if (flt.ne.2) dx(k1)=dx(k1-1)
  go to 210
95 write(nwm,220)imax
220 format(//'  program stop -setup- number of i nodes gt nmax=',i5//)
  stop
```

```
 210 continue
   if (k1.gt.nmax) go to 95
   if (k1.ne.ib2) write(nwm,250)k1
 250 format('  ib2 ne number of nodes, ib2 set = ',i4)
   ib2=k1
   ib1=ib2-1
   ib=ib2-2
   xb(1)=0.
   xc(1)=-dx(1)/2.
   do i=2,ib2
      xb(i)=xb(i-1)+dx(i)
      xc(i)=(xb(i)+xb(i-1))/2.
   end do

   !setup junction area arrays
   k1=0
   do k=1,10
      if (narj(k) .eq. 0) go to 172
      nreg=narj(k)
      arg=arj(k)
      do kk=1,nreg
         if (kk+k1 .gt. nmax) go to 95
         areaj(kk+k1)=arg
      end do
      k1=k1+nreg
   end do
 172 continue
   areaj(k1+1)=areaj(k1)

   !set up the horizontal mesh cell area arrays
   !set up the horizontal mesh cell volum arrays
   k1=1
   do k=1,10
      if (narix(k).eq.0) go to 120
      nreg=narix(k)
      arg=arix(k)
      if (k.eq.1) areaix(1)=arix(1)
      do kk=1,nreg
         if (kk+k1.gt.nmax) go to 95
         volum(kk+k1)=arg*dx(kk+k1)
         areaix(kk+k1)=arg
      end do

     k1=k1+nreg
   end do
 120 continue
   areaix(k1+1)=arg
   if (flb.ne.2) areaix(1)=areaix(2)
   if (flt.ne.2) areaix(ib2)=areaix(ib1)

   !set up the vertical mesh cell area arrays
   !set up the vertical mesh cell volum arrays
   k1=1
   do k=1,10
      if (nariy(k).eq.0) go to 620
      nreg=nariy(k)
      arg=ariy(k)
      if (k.eq.1) areaiy(1)=ariy(1)
      do kk=1,nreg
         if (kk+k1.gt.nmax) go to 95
         volum(kk+k1)=arg*dx(kk+k1)
         areaiy(kk+k1)=arg
      end do
      k1=k1+nreg
   end do
 620 continue
   areaiy(k1+1)=arg
   if (flb.ne.2) areaiy(1)=areaiy(2)
   if (flt.ne.2) areaiy(ib2)=areaiy(ib1)

   !set up the junction cell gravity arrays
   k1=0
   do k=1,10
```

```
      if (ngrav(k).eq.0) go to 420
      nreg=ngrav(k)
      grg=gravo(k)
      do kk=1,nreg
         if (kk+k1.gt.nmax) go to 95
         gravty(kk+k1)=grg
      end do
      k1=k1+nreg
   end do
 420 continue
   gravty(k1+1)=grg
   write(nwm,230)(xb(i),i=1,ib2)
 230 format(/' xb'/10(2x,1pe10.3))
   write(nwm,150)(areaj(i),i=1,ib2)
 150 format(/' areaj'/10(2x,1pe10.3))

   !set up the pressure arrays
   k1=1
   do k=1,10
      if (npo(k).eq.0) go to 2230
      nreg=npo(k)
      pg=po(k)
      do kk=1,nreg
         if (kk+k1.gt.nmax) go to 95
         p(kk+k1)=pg
      end do
      k1=k1+nreg
   end do
 2230 continue

   !set up the initial temperature distribution
   ilast=1
   do k=1,10
      if (ntg(k).gt.0) then
         ifirst=ilast+1
         ilast =ifirst+ntg(k)-1
         do i=ifirst,ilast
            tg(i)=tgo(k)
            tl(i)=tlo(k)
         end do
      else
         goto 520
      endif
   end do
 520 continue
   !set up the gas void fraction arrays
   k1=1
   do k=1,10
      if (nth(k).eq.0) go to 320
      nreg=nth(k)
      thg=tho(k)
      do kk=1,nreg
         if (kk+k1.gt.nmax) go to 95
         th(kk+k1)=thg
      end do
      k1=k1+nreg
   end do
 320 continue

   !set up the initial macrodensity of hydrogen
   k1=1
   do k=1,10
      if (nthh2o(k).eq.0) go to 213
      nreg=nthh2o(k)
      thh2tmp=thh2o(k)
      do kk=1,nreg
         if (kk+k1.gt.nmax) go to 95
         thh2(kk+k1)=thh2tmp
         prh2=p(kk+k1)/((8313./c(130))*tg(kk+k1))
         rh2p(kk+k1)=th(kk+k1)*thh2tmp*prh2
      end do
      k1=k1+nreg
   end do
```

```
213 continue

  !set up the initial macrodensity of non-condensible gas
  k1=1
  do k=1,10
     if (nthncgo(k).eq.0) go to 313
     nreg=nthncgo(k)
     thncgtmp=thncgo(k)
     do kk=1,nreg
        if (kk+k1.gt.nmax) go to 95
        thncg(kk+k1)=thncgtmp
        prncg=p(kk+k1)/((8313./c(137))*tg(kk+k1))
        rncgp(kk+k1)=th(kk+k1)*thncgtmp*prncg
     end do
     k1=k1+nreg  !ttu
  end do
313 continue
  !set up the initial macrodensity of steam  !ttu - added this section
  k1=1
  do k=1,10
     if (nth(k).eq.0) go to 413
     nreg=nth(k)
     do kk=1,nreg
        if (kk+k1.gt.nmax) go to 95
        !prst=p(kk+k1)/(c(12)*tg(kk+k1))
!***************************
  pN2      = p(kk+k1)/1.0D3
  tN2      = tg(kk+k1)

  !assigning initial density within liquid, vapor, and critical region
  if      (tN2.GE.N2TC  .AND.  pN2.GE.N2PC*1000.0) then !above critical
       rhogN2 = N2RHOC !uses critical state for initial
       lN2    = 2
  elseif (tN2.GE.N2TC  .AND.  pN2.LT.N2PC*1000.0) then !vapor phase
       rhogN2 = 0.01   !or uses 44.0 but 0.01 is nearer
       lN2    = 1
  elseif (tN2.LT.N2TC  .AND.  pN2.GT.N2PC*1000.0) then !liquid phase
       rhogN2 = 44.0   !uses liquid phase for initial
       lN2    = 1
  else
       thetaN2 = 1-tN2/N2TC
       psatN2  = N2PC*exp((N2TC/tN2) &
                                        *(N1Psat*thetaN2 &
                                         +N2Psat*(thetaN2**1.5) &
                                         +N3Psat*(thetaN2**2.5) &
                                         +N4Psat*(thetaN2**5)))
       if (pN2 .GT. psatN2*1000.0) then
         rhogN2 = 44.0 !initial liquid phase
         lN2    = 0
         !The highest bound of density observed from N2 table produced by
         !SPAN et al. is 44.0 mol/dm3 in gaseous phase.
       else
         rhogN2 = 0.01 !initial gaseous phase
         lN2    = 1
         !The lowest bound of density observed from N2 table produced by
         !SPAN et al. is 0.01 mol/dm3 in gaseous phase.
       endif
  endif

!In TEXAS, the rg(i) depends on p(i)/R*tg(i) only, even though
!the state is in liquid. Anyhow, this modified module uses the
!equations from Roland Span in the gaseous phase properties in
!the gaseous phase and bounds the gas density by saturation pressure
!inside the liquid phase region.
  if (lN2.EQ.1.OR.lN2.EQ.2) then
       deltaN2  = rhogN2/N2RHOC
       torN2    = N2TC/tN2
       call N2assign
       var0N2   = pN2/(N2RHOC*R*tN2)
       varN2    = deltaN2*(1.0+deltadarbydd())

       do while (dabs(var0N2-varN2).GT.0.0000001)
         dvarN2bydd  = 1+2.0*deltadarbydd()+delta2d2arbydd2()
```

```
          ddeltaN2     = (var0N2-varN2)/dvarN2bydd
          deltaN2      = deltaN2+ddeltaN2
          rhogN2         = N2RHOC*deltaN2
          if (rhogN2.LT.0.01) rhogN2 = 0.01
          if (rhogN2.GT.42.0) rhogN2 = 42.0
          deltaN2      = rhogN2/N2RHOC
          call N2assign
          varN2        = deltaN2*(1.0+deltadarbydd())
        end do

        rhogN2          = deltaN2*N2RHOC              !in mol/dm3
        prst            = rhogN2*N2MolarMass   !in g/dm3 = kg/m3

  elseif (lN2.EQ.0) then
        psat0N2         = p(idummy)/1.0D6
        tsatN2          = N1Psat*N2TC/(log(psat0N2/N2PC)+N1Psat)
        !tsatN2 first approximation
        thetaN2      = 1.0-tsatN2/N2TC
        !first  approximation
        if (thetaN2.LT.0.0 ) then
      tsatN2    = N2TC
          thetaN2       = 0.0 !1.0-tsatN2/N2TC
    endif

    psatN2      = N2PC*exp((N2TC/tsatN2)*(N1Psat*thetaN2 &
                                 +N2Psat*(thetaN2**1.5)+N3Psat*(thetaN2**2.5) &
                                 +N4Psat*(thetaN2**5)))

        do while (dabs(psatN2 - psat0N2).GT.0.000001) !1 Pascal tolerance
          dpsbydtN2=(-1.0)*(N2TC/(tsatN2**2.0))* &
                    (N1Psat*thetaN2     +N2Psat*thetaN2**1.5 &
                          +N3Psat*thetaN2**2.5+N4Psat*thetaN2**5.0) &
                   -(1.0/tsatN2)*(N1Psat+N2Psat*1.5*thetaN2**0.5 &
                          +N3Psat*2.5*thetaN2**1.5+N4Psat*5.0*thetaN2**4.0)
          dpsbydtN2=psatN2*dpsbydtN2

          tsatN2        = (psat0N2-psatN2)/dpsbydtN2+tsatN2
          thetaN2       = 1-tsatN2/N2TC

          psatN2 = N2PC*exp((N2TC/tsatN2)*(N1Psat*thetaN2 &
                               +N2Psat*(thetaN2**1.5)+N3Psat*(thetaN2**2.5) &
                               +N4Psat*(thetaN2**5)))
        end do
        !finding saturated vapor density at p(idummy)
        !the TEXAS should not use the gas_density function
        !if the state is liquid.
        rhogN2 = N2RHOC*exp((N2TC/tsatN2) &
                             *(N1Vsat*(thetaN2**0.34) &
                              +N2Vsat*(thetaN2**(5.0/6.0)) &
                              +N3Vsat*(thetaN2**(7.0/6.0)) &
                              +N4Vsat*(thetaN2**(13.0/6.0)) &
                              +N5Vsat*(thetaN2**(14.0/3.0))))
        prst = rhogN2*N2MolarMass     !in g/dm3 = kg/m3
  endif
!************************
          rgpz(kk+k1)=th(kk+k1)*(1.0-thncg(kk+k1)-thh2(kk+k1))*prst
       end do
       k1=k1+nreg
  end do
413 continue

  !set up the initial fuel particle distribution
  npart=nparn
  if (nparn.eq.0) go to 80
  if (ipopt.ne.0) go to 70
  !uniform distribution between the xpmin and xpmax
  do k=1,npart
     dxp=(xpmax-xpmin)/(npart+1)
     xp(k)=xpmin+k*dxp
  end do
  go to 80
70 continue
  !uniform within each cell
```

```
   npart=ib*nparn
   k=0
   do i=2,ib1
      dxp=dx(i)/(nparn+1)
      do n=1,nparn
         k=1+k
         xp(k)=dxp*n+xb(i-1)
      end do
   end do
 80 continue

   !uniform initial conditions
   p(ib2)=p(ib1)
   tg(ib2)=tg(ib1)
   tl(ib2)=tl(ib1)
   volum(ib2)=volum(ib1)  !ttu - this section
   rgpz(ib2)=rgpz(ib1)
   rncgp(ib2)=rncgp(ib1)
   rh2p(ib2)=rh2p(ib1)
   th(ib2)=th(ib1)
   do i=ib2,2,-1
      call indexf
      rbub(i)=c(47)
      rdrop(i)=c(47)
      ug(i)=ugo
      ul(i)=ulo
      tw(i)=two
      thgas(i)=th(i)
      rg(i)=gas_density(i)     !ttu - this section
      rl(i)=liq_density(i)
      rgp(i)=th(i)*rg(i)
      rlp(i)=(1.-th(i))*rl(i)
      cg(i)=gas_specheat(i)
      ts(i)=gas_tempsat(i)
      siegs(i)=gas_internalenergysat(i)
      sieg(i)=gas_internalenergy(i)
      siel(i)=liq_internalenergy(i)
      call sat(1)
      call mugs
      call muls
      call thercon

      !if pressure distribution is uniform initially, then
      !recalculate pressure distribution considering gravity.
      if (npo(2).eq.0) then
         im=i-1
         dxl=dx(i)+dx(im)
         dxa=dxl/2.
         rmix=rl(i)*(1.-th(im))+rg(i)*th(im)
         p(i-1)=p(i)+gravty(im)*dxa*rmix
      endif
   end do

   ntotal=ngroup+nparn
   if (npart.eq.0) go to 16
   do k=1,npart
      up(k)=upo
      upn(k)=upo
      tp(k)=tpo
      tps(k)=tpo
      tgroup(k)=time
      siep(k)=etot(tp(k))
      rpart(k)=rgrp(igrp(k))
      rpold(k)=rgrp(igrp(k))
      pmas(k)=4*pi*rpart(k)**3*rhop/3
      nump(k)=mppart
      if (ientry.eq.0.or.k.eq.1) then
         ifrag(k)=1
      else
         ifrag(k)=0
      end if
   end do
 16 continue
```

```
  !lower boundary
  i=1
  call indexf
  ug(i)=ug(ip)
  ul(i)=ul(ip)
  tg(i)=tg(ip)
  tl(i)=tl(ip)
  th(i)=th(ip)  !??? shouldn't this be th(i)=thin  ttu
  thncg(i)=thncg(ip)  !ttu added
  thh2(i)=thh2(ip)  !ttu added

  !for inflow bottom(lower) boundary condition, the pressure
  !of boundary will be equal to interior mesh cell pressure
  if (flb.ne.4) p(i)=p(ip)

  !for constant bottom(lower) boundary, the pressure of boundry
  !will not be changed with time; po
  if (flb.eq.4)  p(i)=pin

  !make sure bottom boundary velocities are zero for flb=2.
  if (flb.ne.2) go to 20
  ug(1)=0.
  ul(1)=0.
20 continue
  rbub(i)=c(47)
  rdrop(i)=c(47)
  thn(i)=th(i)
  volum(i)=volum(ip)
  rgpz(i)=(p(i)/p(ip))*rgpz(ip)  !ttu - this section
  rncgp(i)=(p(i)/p(ip))*rncgp(ip)
  rh2p(i)=(p(i)/p(ip))*rh2p(ip)
  ts(i)=gas_tempsat(i)
  rg(i)=gas_density(i)
  rl(i)=liq_density(i)
  rgp(i)=th(i)*rg(i)
  rlp(i)=(1.-th(i))*rl(i)
  cg(i)=gas_specheat(i)
  siegs(i)=gas_internalenergysat(i)
  sieg(i)=gas_internalenergy(i)
  siel(i)=liq_internalenergy(i)
  call sat(1)
  call mugs
  call muls
  call thercon

  !upper boundary !ttu
  i=ib2
  call indexf

  !make sure top boundary velocities are zero for flt=2.
  if (flt.ne.2) go to 25
  ug(ib1)=0.
  ul(ib1)=0.
25 continue

  !the top(upper) boundary pressure is constant; p=pout
  if (flt.eq.4) p(i)=pout

  !free gradient boundary, the top boundary pressure
  !is equal to the interior mesh cell pressure
  if (flt.ne.4) p(i)=p(im)
  tg(i)=tg(i-1)
  tl(i)=tl(i-1)
  th(i)=thout
  thncg(i)=thncg(im)  !ttu added
  thh2(i)=thh2(im)  !ttu added
  rdrop(i)=c(47)
  rbub(i)=c(47)
  thn(i)=th(i)
  volum(i)=volum(im)
  rgpz(i)=(p(i)/p(im))*rgpz(im)  !ttu - this section
  rncgp(i)=(p(i)/p(im))*rncgp(im)
  rh2p(i)=(p(i)/p(im))*rh2p(im)
  ts(i)=gas_tempsat(i)
  rg(i)=gas_density(i)
  rl(i)=liq_density(i)
  rgp(i)=th(i)*rg(i)
  rlp(i)=(1.-th(i))*rl(i)
  cg(i)=gas_specheat(i)
  siegs(i)=gas_internalenergysat(i)
  sieg(i)=gas_internalenergy(i)
  siel(i)=liq_internalenergy(i)
  call sat(1)
  call mugs
  call muls
  call thercon
  call bdry

  !xinj is the fuel jet pouring position.
  xpmr=xpmax-maxsz
  xinj=xb(ib1)
  do i=2,ib1
     if (xb(i-1).le.xpmr.and.xb(i).gt.xpmr) xinj=xb(i-1)
  end do
  if (npart.gt.0) xinj=xb(ib1)

  !calculate the particle fraction distribution
  call thpf

  !calculate the initial masses and energies.
  call masschk
  amassto=amasst
  !esolid--the internal energy for the fuel just about to melt;
  !emelt---the internal energy for the fuel just completely melted;
  !pheat---the fuel latent heat(input);
  !tdelt---introduced delta temperature during fuel melting
  !        to avoid the stright line in the fuel temperature
  !        internal energy curve
!  tdelt=1.
  tdelt=0.01   !urith for ice and water
  tsolid=tmelt-tdelt
  !esolid=c(17)+cp*(tsolid-c(5))
  esolid=c(17)+cp_ice*(tsolid-c(5))
  emelt=esolid+pheat
  tinj=amin1(tijend,tmax)
  npfuelo=1+int(tinj/tpt)
  fuelmo=npin*4./3.*pi*avgsz**3*rhop
  call echk
  efuelo=efuelp
  etotalo=etotal
  egaso=egas
  eliqo=eliq

  !calculate the thermal diffusivity of the fuel
  !which is used to calculate the thermal thickness of the
  !fuel particles
  adiff=kfuel/cp/rhop
end subroutine setupmix

!Original TEXAS
!function hsatf(xdum)
!  include 'fcicom.for'
!  !saturated enthalpy of steam
!  !approx = internal energy plus pressure work
!  !hsatf=siegs(i)+ts(i)*c(12)  !??? shouln't this now be siests(i)?
!  hsatf=siests(i)+ts(i)*c(12)
!end function hsatf

function hsatf(xdum)
  include 'fcicom.for'

  tsatN2 = ts(i)
  if (tsatN2.LE.N2TC) then
    thetaN2 = 1-tsatN2/N2TC
    rhogsatN2 = N2RHOC*exp((N2TC/tsatN2) &
                             *(N1Vsat*(thetaN2**0.34) &
                              +N2Vsat*(thetaN2**(5.0/6.0)) &
                              +N3Vsat*(thetaN2**(7.0/6.0)) &
                              +N4Vsat*(thetaN2**(13.0/6.0)) &
                              +N5Vsat*(thetaN2**(14.0/3.0))))
  else
    tsatN2 = N2TC
    rhogsatN2 = N2RHOC
  endif
  deltaN2 = rhogsatN2/N2RHOC
  torN2   = N2TC/tsatN2
  call N2assign
  hsatf = R*tsatN2*((torda0bydt()+tordarbydt()) &
                    +deltadarbydd()+1.0)                  !in J/mol
  hsatf = hsatf*1000.0/N2MolarMass+ushift                 !in J/kg

end function hsatf

subroutine setc
  include 'fcicom.for'
  !set constants

  ientry2=0
  evaptot=0.0
  fragtot=0.0
  efuelo=0.0
  isiechk=.false.
  idchk=.false.
  idpchk=.false.
  iitchk=.false.
  idtchg=.false.
  iovrid=.false.
  abp=1.0e-8
  absie=1.0e-8 !1.0D5
  alf=1.0
! cp=504.0
  cp=4.179e3    !J/kg in TEXAS for both water and ice from
                !Heat Transfer,J.P.Holman,page 642
  dxvac=0.00025 !vacuum cell length
  dtmax=10.
  dtmin=1.0e-9
  epsg=0.01
  epsd=0.2
  epsi=0.5
  epsl=0.01
  epsp=5.0
  flt=3
  flb=2
  ib=8
  itmax=100
  itup=5
  kb=1.38e-23  !Boltzmann constant (J/K)
  knconvac=0.1 !Knudsen number: above=vacuum, below=continuum
  maxn=50
  maxprt=25
  mpart=100
  ndiv=5
  nvar=19
  pmass=4.6e-5
  pi=3.14159
  pin=1.0e5
  pvcratio=0.75 !Fraction of continuum press where vac switches to con
! rhop=10.97e3
  rhop=0.910e3 !kg/m3
  rparn=1.0e-3
  thpmx=0.61
  thstar=0.5
! tgin=373.0
  tgin=77.244  !Kelvin
! tlin=373.0
  tlin=77.244  !Kelvin
  tpin=373.0
  tpo=373.0
  tpt=1.0e6
! two=373.0
  two=77.244   !Kelvin
  xpmax=1.0
  ifragmix=1
  nbottom=10000000
  cycheck=1000000
  dtcheck=-1.
  nbreak=1
  nfront=2
  N2MolarMass=28.01348
  ushift = 1.0e6 !J/kg
  !ushift=4222.8*1000.0/N2MolarMass is 150.7e3 J/kg
  !150.7e3 J/kg is not enough to shift ice at 77.244K
  !u of ice at 77.244K = -333.43e3-1.93e3*(273.15-77.244)
  !                    = -711.5e3 J/kg
  !select shift the spec.internal eng. by 1.0e6J/kg
  c(1)=-4797.9     !1st coefficient of sat vap temp function
  c(2)=1.          !2nd coefficient of sat vap temp function
  c(3)=-12.8576    !3rd coefficient of sat vap temp function
! c(4)=1.4882e-3   !molten fuel's viscosity
  c(4)=1.75e-3    !N.s/m2 @ 273K for H2O from Fox & McDonald
! c(5)=373.        !reference temperature
  c(5)=77.244     !for N2
! c(6)=2.509e6     !vapor specific internal energy at reference temperature
  c(6)=1544.3 *1000.0/28.01348 + ushift      !J/kg for N2
! c(7)=4.19e3      !liquid specific heat
  c(7)=57.17  *1000.0/28.01348!J/kg-K for N2
! c(8)=0.419e6     !liquid specific internal energy at reference temperature
  c(8)=-3428.0*1000.0/28.01348 + ushift      !J/kg for N2
! c(9)=0.94e3      !microscopic density of liquid
  c(9)=28.793*28.01348
! c(10)=2257.e3    !latent heat constant
  c(10)=(2159.1-(-3424.6))*1000.0/28.01348  !J/kg for N2 (hfg=enthalpy fg)
  c(11)=1e5        !reference pressure
! c(12)=4.62e2     !gas constant for vapor
! c(12) used in sub hsatf, func gas_density, func gas_recipsoundspeedsq,
! sub setupmix only
  c(12)=0.0           !value of zero is assigned
! c(13)=2796.e-5   !vapor thermal conductivity
! c(13)=0.026      !N2 vapor thermal conductivity from CRC handbook of tables
                   !for applied engineering science,page 48
  c(13)=0.00415*1.7307 !N2 vapor thermal conductivity from CRC handbook of
                   !tables for applied engineering science,page 97
! c(14)=6.8e-1     !liquid thermal conductivity
  c(14)=0.0804*1.7307  !N2 liquid thermal conductivity from CRC handbook of
                      !tables for applied engineering science,page 97
                      !0.13915
! c(15)=0.662e-6   !square of liquid reciprocal sound speed
  c(15)=(1.0/852.5)**2.0 !Roland Span et al.
! c(16)=1.3e0      !gamma, the ratio of vapor specific heats (cp/cv)
  c(16)=1.4e0      !assumed at STP from Fox & McDonald
! c(17)=3.78e3     !particle internal energy at reference temperature
  c(17)=-742.8e3 + ushift     !J/kg derived from cp=1.93e3J/kg,etot=-333.43e3J/kg,
                              !t=273.15K and tref=77.244K
! c(18)=7.8e-1     !radiation emissivity of particle
  c(18)=9.5e-1     !from TEXAS water
! c(19)=9.5e-1     !radiation emissivity of water
  c(19)=9.5e-1     !value assigned such that at low temperature
                   !the radiation is very little effect
  c(20)=5e-1       !vol frac criterion of vapor & liquid internal energy
!                   updated by adding viscous work and ht conduction
  c(21)=1e0        !pressure expansion work calc out/in (0/1) press iter loop
! c(22)=7.4e-4     !thermal expansion coefficient of water.
  c(22)=5.6e-3     !Liquid N2 derived data from Roland Span et al. @0.1MPa,
                   !(28.793mol/dm3,77.244K) and (29.155mol/dm3,75K)
! c(23)=2.8e-4     !liquid viscosity @ 373K
  c(23)=0.382*4.1338e-4 !kg/m-s N2 from CRC handbook of
                  !tables for applied engineering science,page 97
! c(24)=72.e-3     !surface tension
  c(24)=6.2e-3     !N2 liquid surface tension at -183 C
                  !from Physical Chemistry by Silbey and Alberty,page 187
! c(25)=1.3e-5    !vapor viscosity @ 373K
```

```
  c(25)=0.0134*4.1338e-4      !kg/m-s N2 from CRC handbook of
                              !tables for applied engineering science,page 97
  c(26)=5.67e-8     !Stefan-Boltmann constant
  c(27)=11.0        !critical weber number of bubble
  c(28)=12.0        !critical weber number of droplet
! c(29)=7.0         !particle thermal conductivity
  c(29)=7.5         !Ice thermal conductivity @ 100K from CRC handbook of
                    !tables for applied engineering science,page 185
  c(31)=12.0        !critical weber number of fuel particle
! c(32)=0.5         !particle surface tension
  c(32)=0.0757      !N/m @ 273K for H2O from Fox & McDonald
  c(39)=1e0         !convective energy flux calc out/in (0/1) press iter loop
  c(41)=0.1         !fraction of vap ht. cap. of vap existing in vap film
  c(42)=1e-4        !vol frac of s dispersed phase for min drag coeff
  c(45)=0.1         !vapor bubble thermal thickness
  c(47)=1D-3        !minimum siae of the bubble or droplet.
  c(49)=0.1093      !first coeff in time-independent linear frag model
  c(50)=-0.0785     !second coeff in time-independent linear frag model
  c(51)=1.0         !third coeff in time-independent linear frag model
  c(52)=0.246       !fourth coeff in time-independent linear frag model
! c(59)=0.1         !radiation emissivity of vapor
  c(59)=0.1         !value assigned such that at low temperature
                    !the radiation is very little effect
  c(103)=2.727e8    !exothermic combustion energy of metal w/ stm (Zr here)
  c(105)=91.22      !molecular weight of reacting metal (Zr here)
  c(106)=2.0        !molar ratio of h2 to metal in oxidizing eq (Zr here)
  c(119)=9.9e8      !minimum pressure at which valve would remain open
  c(120)=9.9e8      !minimum pressure at which valve would remain closed

  c(130)=2.016      ! Molecular weight of hydrogen
  !values for H2 below from SPWANG - need to change spec heat (ttu)
  c(131)=1.0e-5     ! Viscosity of hydrogen
  c(132)=0.226      ! Thermal conductivity of hydrogen
  c(133)=1.443e4    ! Specific heat of hydrogen
  c(134)=4.94e6     ! Reference internal energy of hydrogen
  c(135)=350.0      ! Reference temperature of hydrogen
  c(136)=0.0        ! Fraction of oxidized metal during explosion
  c(137)=28.966     ! Molecular weight of ncg (air value used)
  c(138)=1.589e-5   ! Viscosity of ncg (air value at 300K used)
  c(139)=0.0263     ! Thermal conductivity of ncg (air value at 300K used)
  c(140)=3.0e-10    ! Molecular diameter of ncg (water molecule value used)

  c(144)=719.       ! Specific heat of ncg (air value at 300K used)
  c(145)=2.16e5     ! Reference internal energy of ncg (air value at 300K used)
  c(146)=300.0      ! Reference temperature of ncg
  c(147)=1.34       ! Specific heat ratio for steam
  c(148)=1.407      ! Specific heat ratio for h2 (noble gas value used)
  c(149)=1.404      ! Specific heat ratio for ncg (air value used)
  c(141)=0.01       ! New-Frag Coeff for KH process
  c(142)=0.2        ! Thick/Thin void fraction criteria
  c(143)=1e6        ! max number of particles in one group
  c(150)=0.0        ! check r/lambda < 10 for KH (0=No check)

  brktp=0.5         ! half sphere oxidation type
  ih2gen1=1         ! using steam diffusion controled model
  ih2gen2=2         ! identical volume transfer of oxide layer
  ih2gen3=1         ! original surface area is used in kinetic law
  mmetal=2          ! coefficient in kinetic rate law for Zr
  deltam0=0.0       ! initial oxide layer thickness

  !constant for eos of nitrogen at saturation
  !From Roland Span, Eric W.Lemmon, Richard T Jacobsen,
  !Wolfgang Wagner, Akimichi Yokozeki
  !"A Reference Equation of State for the Thermodynamics Properties
  !of Nitrogen for Temperatures from 63.151 to 1000 K and Pressures to
  !2200 MPa
  R       =8.314510
  N2TC    =126.192                     !Kelvin
  N2PC    =  3.3958                    !Mega pascal
  N21PC   =  3.3958D6                  !pascal
  N2RHOC  = 11.1839                    !mol/dm3
  N21RHOC = 11.1839*28.01348           !kg/m3
```

```
N1Psat=-6.12445284
N2Psat=1.26327220
N3Psat=-0.765910082
N4Psat=-1.77570564

N1Lsat=1.48654237
N2Lsat=-0.280476066
N3Lsat=0.0894143085
N4Lsat=-0.119879866

N1Vsat=-1.70127164
N2Vsat=-3.70402649
N3Vsat=1.29859383
N4Vsat=-0.561424977
N5Vsat=-2.68505381

N2N(1) = 0.924803575275D0
N2N(2) =-0.492448489428D0
N2N(3) = 0.661883336938D0
N2N(4) =-0.192902649201D1
N2N(5) =-0.622469309629D-1
N2N(6) = 0.349943957581D0
N2N(7) = 0.564857472498D0
N2N(8) =-0.161720005987D1
N2N(9) =-0.481395031883D0
N2N(10)= 0.421150636384D0
N2N(11)=-0.161962230825D-1
N2N(12)= 0.172100994165D0
N2N(13)= 0.735448924933D-2
N2N(14)= 0.168077305479D-1
N2N(15)=-0.107626664179D-2
N2N(16)=-0.137318088513D-1
N2N(17)= 0.635466899859D-3
N2N(18)= 0.304432279419D-2
N2N(19)=-0.435762336045D-1
N2N(20)=-0.723174889316D-1
N2N(21)= 0.389644315272D-1
N2N(22)=-0.212201363910D-1
N2N(23)= 0.408822981509D-2
N2N(24)=-0.551990017984D-4
N2N(25)=-0.462016716479D-1
N2N(26)=-0.300311716011D-2
N2N(27)= 0.368825891208D-1
N2N(28)=-0.255856846220D-2
N2N(29)= 0.896915264558D-2
N2N(30)=-0.441513370350D-2
N2N(31)= 0.133722924858D-2
N2N(32)= 0.264832491957D-3
N2N(33)= 0.196688194015D2
N2N(34)=-0.209115600730D2
N2N(35)= 0.167788306989D-1
N2N(36)= 0.262767566274D4

N2i(1) =1.0
N2i(2) =1.0
N2i(3) =2.0
N2i(4) =2.0
N2i(5) =3.0
N2i(6) =3.0
N2i(7) =1.0
N2i(8) =1.0
N2i(9) =1.0
N2i(10)=3.0
N2i(11)=3.0
N2i(12)=4.0
N2i(13)=6.0
N2i(14)=6.0
N2i(15)=7.0
N2i(16)=7.0
N2i(17)=8.0
N2i(18)=8.0
N2i(19)=1.0
N2i(20)=2.0
```

```
N2i(21)=3.0
N2i(22)=4.0
N2i(23)=5.0
N2i(24)=8.0
N2i(25)=4.0
N2i(26)=5.0
N2i(27)=5.0
N2i(28)=8.0
N2i(29)=3.0
N2i(30)=5.0
N2i(31)=6.0
N2i(32)=9.0
N2i(33)=1.0
N2i(34)=1.0
N2i(35)=3.0
N2i(36)=2.0

N2j(1)  = 0.25
N2j(2)  = 0.875
N2j(3)  = 0.5
N2j(4)  = 0.875
N2j(5)  = 0.375
N2j(6)  = 0.75
N2j(7)  = 0.5
N2j(8)  = 0.75
N2j(9)  = 2.0
N2j(10)= 1.25
N2j(11)= 3.5
N2j(12)= 1.0
N2j(13)= 0.5
N2j(14)= 3.0
N2j(15)= 0.0
N2j(16)= 2.75
N2j(17)= 0.75
N2j(18)= 2.5
N2j(19)= 4.0
N2j(20)= 6.0
N2j(21)= 6.0
N2j(22)= 3.0
N2j(23)= 3.0
N2j(24)= 6.0
N2j(25)=16.0
N2j(26)=11.0
N2j(27)=15.0
N2j(28)=12.0
N2j(29)=12.0
N2j(30)= 7.0
N2j(31)= 4.0
N2j(32)=16.0
N2j(33)= 0.0
N2j(34)= 1.0
N2j(35)= 2.0
N2j(36)= 3.0

N2l(1)  =0
N2l(2)  =0
N2l(3)  =0
N2l(4)  =0
N2l(5)  =0
N2l(6)  =0
N2l(7)  =1
N2l(8)  =1
N2l(9)  =1
N2l(10)=1
N2l(11)=1
N2l(12)=1
N2l(13)=1
N2l(14)=1
N2l(15)=1
N2l(16)=1
N2l(17)=1
N2l(18)=1
N2l(19)=2
```

```
    N2l(20)=2
    N2l(21)=2
    N2l(22)=2
    N2l(23)=2
    N2l(24)=2
    N2l(25)=3
    N2l(26)=3
    N2l(27)=3
    N2l(28)=3
    N2l(29)=4
    N2l(30)=4
    N2l(31)=4
    N2l(32)=4
    N2l(33)=2
    N2l(34)=2
    N2l(35)=2
    N2l(36)=2

    N2phi(33)=20
    N2phi(34)=20
    N2phi(35)=15
    N2phi(36)=25

    N2beta(33)=325
    N2beta(34)=325
    N2beta(35)=300
    N2beta(36)=275

    N2gamma(33)=1.16
    N2gamma(34)=1.16
    N2gamma(35)=1.13
    N2gamma(36)=1.25

    N2a(1)=2.5
    N2a(2)=-12.76952708
    N2a(3)=-0.00784163
    N2a(4)=-1.934819D-4
    N2a(5)=-1.247742D-5
    N2a(6)=6.678326D-8
    N2a(7)=1.012941
    N2a(8)=26.65788

  end subroutine setc

  subroutine qwall
   include 'fcicom.for'
    !calculate the heat transfer from the continuous phase
    !to the wall of each mesh cell by convective heat trans

    qlw(i)=0.0
    qgw(i)=0.0
    wlw(i)=0.0
    wgw(i)=0.0
    !do not consider the heat transfer to the wall temporarily
    !iay=1
    !if (iay.eq.0) return ! a modification for hp9750
    if (c(160).eq.0) return

    dhy=2.*(volum(i)/(pi*dx(i)))**0.5
    !to calculate the contact area between the continuous
    !phase and the wall due to some fuel contact the wall.
    awall=(4.*pi*dx(i)*volum(i))**0.5
    if (npart.eq.0) go to 11
    do k=1,npart
       if (xp(k).lt.xb(i-1).or.xp(k).ge.xb(i)) go to 10
       !calculate the contact area between fuel and wall
       call areap
       awall=awall-nump(k)*areapw
10     continue
    end do
11 continue
    !calculate the reynold no and prandtl no of single phase
    prg=c(25)*cg(i)/kapg(i)
  prl=c(23)*c(7)/kapl(i)
  reg=rgn(i)*abs(ug(i)+ug(i-1))*0.5*dhy/c(25)
  rel=rln(i)*abs(ul(i)+ul(i-1))*0.5*dhy/c(23)
  !in bubbly flow, the heat transfer coef. between wall
  !and liquid will be the maximum value of turbulent
  !and laminar convective coef. at constant wall temp.
  if (imap(i)..eq.1) then
     !turbulent ht-trans coef for flow in the tube
     hltur=0.021*prl**0.5*(rel*(1.-th(i)))**0.8 &
          *kapl(i)/(dhy*(1.-th(i)))
     !laminar ht-trans coef for flow in the tube with constant tw
     hllam=3.658*kapl(i)/(dhy*(1.-th(i)))
     hlw=amax1(hllam,hltur)
         thlx=amax1(1.0-th(i),1.0-1.0e-5)
         hlw=hlw/(volum(i)*thlx)       !Sunchai and Urith
     qlw(i)=hlw*awall
     wlw(i)=qlw(i)*tw(i)
     !in droplet(mist) flow, the heat trans coef. between vapor
     !and wall will be the maximum value of the turbulent and
     !laminar convective heat trans coef. at constant wall temp
  else if (imap(i).eq.3) then
     hgtur=0.021*prg**0.5*(reg*th(i))**0.8*kapg(i)/(dhy*th(i))
     hglam=3.658*kapg(i)/(dhy*th(i))
     hgw=amax1(hgtur,hglam)
         thgx=amax1(th(i),1e-5)
         hgw=hgw/(volum(i)*thgx)       !Sunchai and Urith
     qgw(i)=hgw*awall
     wgw(i)=qgw(i)*tw(i)

     !in the transition flow the heat trans coef. from the flow
     !to the wall will be a combination of bubbly and mist flow.
  else if (imap(i).eq.2) then
     hltur=0.021*prl**0.5*(rel*(1.-th(i)))**0.8 &
          *kapl(i)/(dhy*(1.-th(i)))
     hllam=3.658*kapl(i)/(dhy*(1.-th(i)))
     hlw=amax1(hltur,hllam)
         thlx=amax1(1.0-th(i),1.0-1.0e-5)
         hlw=hlw/(volum(i)*thlx)       !Sunchai and Urith
     hgtur=0.021*prg**0.5*(reg*th(i))**0.8*kapg(i)/(dhy*th(i))
     hglam=3.658*kapg(i)/(dhy*th(i))
     hgw=amax1(hgtur,hglam)
         thgx=amax1(th(i),1e-5)
         hgw=hgw/(volum(i)*thgx)       !Sunchai and Urith
     !in the transition flow, there is liquid or vapor continuous
     !phase.  hence the contact area between wall and each
     !continuous phase depends on their void frac in this mesh cell
     qlw(i)=hlw*awall*(1.-th(i)+thgl(i)-thlg(i))
     wlw(i)=qlw(i)*tw(i)
     qgw(i)=hgw*awall*(th(i)+thlg(i)-thgl(i))
     wgw(i)=qgw(i)*tw(i)
  endif

end subroutine qwall

subroutine evpcod
  include 'fcicom.for'
  !calculate condensation and evaporation rate

  f(x,y)=c(56)/pi*(-atan(100.*(x-y))+pi/2.) !test.in c(56)=1.0
  erate(i)=0.0
  crate(i)=0.0
  efuel=0.0
  erad=0.0
  !if liquid temperature tl smaller than the saturation temp
  !then there is a portion of radiation heat transfer from the
  !fuel to the liquid-vapor interface will be transfered to
  !the liquid field,i.e., irad*qrad and 0.5<irad<1.0

  !on the other hand, if tl > ts then the whole radiation heat
  !is used to evaporate the steam.

  if (c(107).le.0.0) irad(i)=f(tl(i),ts(i)) !test.in c(107)=1
```

```
   !bubble flow regime
   if (imap(i).eq.1) then
      !part(i): fuel - vapor film - bulk liquid
      !heat trans from fuel to liquid-vapor interface substract
      !heat trans from interface to bulk liquid will evaporate
      !or condensate depending on positive or negative of net heat
      if (npart.eq.0.) go to 26
      do k=1,npart
         if (thpsv(i,k).eq.0.0) go to 25
         call areap
         hfgbar=1+c(41)*cg(i)*(tps(k)-ts(i))/lheat(i)
         e1=areapl*(hffilm(k)*(tps(k)-ts(i))-hconv1(k) &
              *(ts(i)-tl(i)))*nump(k)/(volum(i)*hfgbar)       !Watt/volume
         !if the net heat from the particles to the vapor-liquid
         !interface is less than zero, set it to zero.
         !after the fuel particle starts quenching, there is no
         !vapor generated and all the heat goes to bulk liquid.
         if (e1.lt.0. .or. iquench(k).gt.0) e1=0.
         efuel=efuel+e1*thpsv(i,k)
25       continue
      end do
26    continue
      !part(2): vapor(bubble) - bulk liquid
      !evaporatiton or condensation depends on the positive or
      !negative of the net heat flow from vapor(bubble) to
      !vapor-liquid interface substracts interface to liquid
      ebub=qgsg(i)*(tg(i)-ts(i))-qlsl(i)*(ts(i)-tl(i))
         ebub=ebub+qlw(i)*(tw(i)-tl(i))                !Urith
      htrate=efuel+ebub

   !transition flow regime
   else if (imap(i).eq.2) then
      !include all heat transfer path, from bubble, vapor film
      !(or fuel) and bulk vapor to the l-v interface and from the
      !interface to the bulk liquid and liquid droplet.
      if (npart.eq.0.) go to 36
      do k=1,npart
         if (thpsv(i,k).eq.0.0) go to 35
         call areap
         hfgbar=1+c(41)*cg(i)*(tps(k)-ts(i))/lheat(i)
         e1=areapl*(hffilm(k)*(tps(k)-ts(i))-hconv1(k) &
              *(ts(i)-tl(i)))*nump(k)/(volum(i)*hfgbar)
         !if the net heat from the particles to the vapor-liquid
         !interface is less than zero, set it to zero.
         !after the fuel particle starts quenching, there is no
         !vapor generated and all the heat goes to bulk liquid.
         if (e1.lt.0. .or. iquench(k).gt.0 ) e1=0.
         efuel=efuel+e1*thpsv(i,k)
35       continue
      end do
36    continue
      etran=qgsg(i)*(tg(i)-ts(i))-qlsl(i)*(ts(i)-tl(i))
         etran=etran+qlw(i)*(tw(i)-tl(i))+qgw(i)*(tw(i)-tg(i))  !Urith
      htrate=efuel+etran

   !mist flow regime convective heat trans is dominant
   else if (imap(i).eq.3) then
      edrop=qgsg(i)*(tg(i)-ts(i))-qlsl(i)*(ts(i)-tl(i))
         edrop=edrop+qgw(i)*(tw(i)-tg(i))              !Urith
      htrate=edrop
   end if

   !if liquid temp greater than the saturation temp, the
   !radition heat trans from fuel to the liquid will evaprate
   !spontaneously, otherwise the radiation heat will be
   !absorbed in the liquid phase.
   if (npart.eq.0.) go to 56
   do k=1,npart
      if (thpsv(i,k).eq.0.0)  go to 55
      call areap
      if (c(107).gt.0.0) irad(i)=xirad(i,k)
      hfgbar=1+c(41)*cg(i)*(tps(k)-ts(i))/lheat(i)
      er1=areapl*hrad(k)*(tps(k)-ts(i))*nump(k)/(volum(i)*hfgbar)
      er1=er1*(1-irad(i))
      er2=areapg*hradpl(k)*(tps(k)-ts(i))*nump(k)/volum(i)
      er2=er2*(1-irad(i))
      if (iquench(k).gt.0.) then
         er1=0.
         er2=0.
      endif
      erad=erad+(er1+er2)*thpsv(i,k)
55    continue
   end do
56 continue

   !estimate the net mass transfer in the mesh cell (i)
   !ecrate > 0,  evaporation
   !ecrate < 0,  condensation

   ecrate=(htrate+erad+evap(i))/lheat(i)/(1.-thp(i))
!   if (thp(i).lt.1.e-7) ecrate=0.                              !Urith
!   if (ecrate.ge.0.) erate(i)=ecrate                           !Urith
!   if (ecrate.lt.0.) crate(i)=abs(ecrate)                      !Urith
   if (ecrate.ge.0.0) then                                      !Urith
     erate(i)=ecrate                                            !Urith
        if (rlp(i).eq.0.0) then                                 !Urith
           erate(i)=0.0                                         !Urith
        else                                                    !Urith
           if (erate(i)*dt.gt.rlp(i)) erate(i)=rlp(i)/dt        !Urith
        endif                                                   !Urith
   endif                                                        !Urith
   if (ecrate.lt.0.) then                                       !Urith
     crate(i)=abs(ecrate)                                       !Urith
        if (rgp(i).eq.0.0) then                                 !Urith
           crate(i)=0.0                                         !Urith
        else                                                    !Urith
           if (crate(i)*dt.gt.rgp(i)) crate(i)=rgp(i)/dt        !Urith
        endif                                                   !Urith
   endif                                                        !Urith

end subroutine evpcod

subroutine heatlg
   include 'fcicom.for'
   !liquid-vapor heat transfer coeffcient between
   !bubble and bulk liquid in the bubble regime or
   !liquid drop and continuous vapor phase


   !calculate liquid-vapor interfacial area of mesh cell (i)
   call arelg

   !INTERFACE does not exist if th(i)=0 or th(i)=1    !Urith
   thmin=(4./3.)*pi*c(47)**3.0/(volum(i)*(1.0-thp(i)))          !Urith
   if (th(i).lt.thmin.or.th(i).gt.1.0-thmin) then               !Urith
         qgsg(i)=0.0
         qlsl(i)=0.0
         return
   end if

   !htot1: heat trans coef between vapor bubble and interface
   !htot2: heat trans coef between vapor and drop interface
   !htot3: heat trans coef between vapor and regime interface
   !htot4: heat trans coef between liquid drop and interface
   !htot5: heat trans coef between bubble interface and liquid
   !htot6: heat trans coef between liquid and regime interface

   htot1=kapg(i)*5./rbub(i)
   htot2=hconvg(i)
   htot3=hgl(i)
   htot4=kapl(i)*5./rdrop(i)
   htot5=hconv2(i)
   htot6=hlg(i)

   !adjust the liquid-side thermal thickness of vapor bubble
   !due to the moving boundary by high mass transfer rate.
```

```
  const1=dt*kapl(i)*(ts(i)-tl(i))/(rg(i)*(lheat(i)+cg(i)* &
       (tg(i)-ts(i))))

  !update liquid-side heat transfer coefficient.
  if (const1.gt.0.0) then
     htot5=htot5/(1.+2*const1*(htot5/kapl(i))**2)**0.5
  end if

  !adjust the thermal  thickness of vapor bubble
  !at the high mass transfer rate, especially, condensation.
  deltab=c(45)*rbub(i)
  if (c(45).ne.0.) then
     htot1=kapg(i)/deltab
     cs1=htot5*(ts(i)-tl(i))
     cs2=htot1*(tg(i)-ts(i))
     if (cs1.eq.cs2.or.cs2.eq.0.) go to 100
     cs3=lheat(i)*(1+cs1/(cs2-cs1))
     if (cs3.eq.0.) go to 100
     rtemp=cg(i)*(tg(i)-ts(i))/cs3

     !evaporation case
     if (rtemp.gt.0.) then
        rtemp=amax1(rtemp,1e-3)
        rtemp=amin1(rtemp,2e0)

     !condensation case
     else
        rtemp=amin1(rtemp,-1e-3)
        rtemp=amax1(rtemp,-9.99e-1)
     end if

     !update thermal thickness
     deltab=deltab*rtemp/log(rtemp+1)
     deltab=amin1(5e-1*rbub(i),deltab)
     deltab=amax1(1e-3*rbub(i),deltab)

     !update the vapor-side heat transfer coef in the bubble
100  continue
     htot1=kapg(i)/deltab
  end if

  !calculate the macroscopic ht trans coef between liquid,
  !vapor and interface qlsl, qgsg repectively.
  if (imap(i).eq.1) then
     qgsg(i)=htot1*areag
     qlsl(i)=htot5*areag
  else if (imap(i).eq.2) then
     qgsg(i)=(htot1*areag+htot2*areal+htot3*arealg)
     qlsl(i)=(htot5*areag+htot4*areal+htot6*arealg)
  else
     qgsg(i)=htot2*areal
     qlsl(i)=htot4*areal
  end if

end subroutine heatlg

program  main
  include 'fcicom.for'
  dimension cs(4)
  character headin(16)*60, disclm(8)*60
  data (headin(i),i=1,16)/ &
  'c**********************************************************c', &
  'c**********************************************************c', &
  'c****                                                  ****c', &
  'c****                    texas-V                       ****c', &
  'c****        fuel-coolant fragmentation & mixing       ****c', &
  'c****                    S P Wang                      ****c', &
  'c****                  Tris Utschig                    ****c', &
  'c****              Sunchai Nilsuwankosit               ****c', &
  'c****                   jian tang                      ****c', &
  'c****                 cho-chone c. chu                 ****c', &
  'c****           nuclear engineering department         ****c', &
  'c****           university of wisconsin-madison        ****c', &
  'c****             madison, wisconsin 53706            ****c', &
  'c****                                                 ****c', &
  'c*********************************************************c', &
  'c*********************************************************c'/
  data (disclm(i),i=1,8)/ &
  'c*********************************************************c', &
  '                  dec. 10, 1991                           ', &
  '   the texas code is an experimental 3-field eulerian-    ', &
  'lagrangian 1d hydrodynamics model designed specifically for ', &
  'calculation of fcis. this version of texas may still contain', &
  'bugs. it is being released to friendly users for evaluation.', &
  'further dissemination of the code is forbidden.           ', &
  'c*********************************************************c'/

  !INPUT/OUTPUT SETUP
  call input      !check and read input
  call fileopen   !open output files
  write(nwm,302) (headin(i)(1:60),i=1,16)
  write(nwm,302) (disclm(i)(1:60),i=1,8)
302 format(/(1x,a60))
  write (*,*) 'fileopen pass'
  if (nfront.lt.2) nfront=2
  if (nfront.gt.ngmax) nfront=ngmax
  ninjet=nparn
  if ((c(98).eq.0.).and.(tijend.gt.0.0)) call adjgrp
  write (*,*) 'adjgrp pass'
  !INITIALIZE PROBLEM
  if (irest.ne.2) then
     if (iexplos.eq.0) call setupmix
        write (*,*) 'setupmix pass'
     if (iexplos.eq.1) call setupexp
     call inputsetup  !write out initialized data
     !initialize time variables
     imax=nmax
     tpli=tpl+time;  tplo15=tpl15+time;  tplo99=tpl99+time
     tpart=time;  tdump=dtdump
     !start of time loop
     nit=0
     !if irest=2, restart calculation from last dump
  else
     call restart
     close(unit=50)
  endif
  write (*,*) 'initialize pass'
  !TOP OF TIME STEP LOOP
170 continue
  !dump all necessary info for restart at specified time interval.
  if (irest.ne.1) then
     if (time.gt.tdump.or.time.gt.tmax) then
        open(unit=50,file='restfile',form='unformatted', &
             status='unknown')
        call dump
        close(unit=50)
        tdump=tdump+dtdump
     endif
  endif
  !end calculation if time beyond max
  if (time.ge.tmax+dt) go to 190

  call bdry                         !adjust boundary condition
  call oldnew                       !save old properties for next time step
  if ((iexplos.eq.0) &
       .and.(tijend.gt.0.0)) then   !ttu added this part of if statement
     call inject                    !inject particles
  end if
  call masschk                      !check mass conservation
  call echk                         !check energy conservation
  call h2track                      !track hydrogen
  call print                        !output
  call dtset                        !adjust time step
  call poldnew                      !save particle props before press. iter.
!  call vacfind                      !check for vacuum flow
```

```
  !ITERATION FOR PRESSURE BEGINS
230 continue

  call pnewold !reset particle props to those before pressure iteration
  call tcontrl !adjust timestep if needed
  call newold  !reset gas/liq  props to those before pressure iteration

  !PARTICLE DYNAMICS
  if (ifragmix.eq.1) call newfrag   !explicitly estimate fragmentation
  if (iexplos.eq.1) call expfrg     !fuel size by hydrodynamic frag model
  call upart                        !advance particle positions
  call thpf                         !calculate particle distribution
  call thpave
  call vacfind
  !HEAT TRANSFER
  do i=2,ib1
        call flowmp     !find out flow regime
     call rbubdp     !calculate bubble & droplet radius
     call htcolg     !calculate liq-vap ht tr coef
     if (npart.eq.0.) go to 101
     do k=1,npart
        if (xp(k).lt.xb(i-1) &
            .or.xp(k).ge.xb(i)) &      !calculate fuel-liq/vap ht tr coef
            go to 100
        call htcop
100     continue
     end do
101  continue
     call heatlg     !calc vap-interface, interface-liq macro ht tr coefs
!     call evpcod     !calc evaporation or condesation rate
!     call qwall      !calc wall-liq or wall-vap ht tr coef
     call qwall         !Urith and Sunchai
        call evpcod      !Urith and Sunchai
  end do

  !DRAG
  do i=1,ib1
     call kdrags   !estimate vap-liq drag coef with old time void
     call kwalls   !and relative velocity
  end do
  do i=1,ib2       !estimate liq-vap virtual mass force
     call vmassf   !from change of relative velocity
  end do

  !VELOCITIES
  call velt        !calc new vels w/out fuel momentum exchange (t level)
  call velt2       !calc new vels w/ fuel momentum exchange (2t level)
  do i=2,ib1
     call indexf   !recalc new vels w/ updated fuel momentum exchange
     call vels2    !but old convection momentum;, pressure gradient; mass
  end do           !transfer; and viscous, drag, and virtual mass forces

  !PRESSURE ITERATION
  do i=2,ib1                        !set up beta(j,i) matrix
     if (th(i).gt.thflag) ithf(i)=1 !compare vapor void fraction with
     if (th(i).lt.thflag) ithf(i)=0 !specified void frac (usually ~ 0.5)
  end do                            !if th(i)<thstar beta(j,i)=ddl(i)/dp(j)
  idtchg=.false.                    !if th(i)>thstar beta(j,i)=ddg(i)/dp(j)
!call vacfind
  call iter2                        !perform pressure iteration
  call itchk                        !check if iteration no. over limit
  if (iitchk) then
!    write(*,*)'iitchk ',iitchk
    go to 230              !if iteration no. over limit reduce dt
  endif
  if (isiechk) then
!       write(*,*)'isiechk ',isiechk
    go to 230             !if intenrgy change over limit reduce dt
  endif
  if (idpchk) then
!       write(*,*)'idpchk ',idpchk
    go to 230              !if pressure change over limit reduce dt
  endif
  if (idchk) then
!       write(*,*)'idchk ',idchk
    go to 230               !if error function over limit reduce dt
  endif

  !SOLUTION CONVERGED, COMPLETE UPDATING of internal energies
  !add pressure compression, conduction and viscous work term
  do i=2,ib1
     call indexf
     call thercon
     call heats
     call vworkl; call vworkg

     !ADD CONVECTION ENTHALPY term by user's option
     if (c(39).ne.0.) then
        fel(i)=0.0; feg(i)=0.0          !enthalpy flux in iteration loop
     else
        call sielf; call siegf          !enthalpy flux out of iteration loop
     end if
     !ADD INTER-PHASE PRESSURE WORK term by user's option
     if (c(21).ne.0.) then
        cs(1)=0.0; cs(2)=0.0            !pressure work in iteration loop
     else
        cs(1)=p(i)*((th(i)-thn(i))/dt)  !pressure work out of iteration loop
        cs(2)=p(i)*(-(th(i)-thn(i))/dt)
     end if
     cs(3)=rlp(i)*siel(i)+dt*(-cs(2)+workl+heatl-fel(i))
     cs(4)=rgp(i)*sieg(i)+dt*(-cs(1)+workg+heatg-feg(i))
     if ((1.-th(i)).gt.c(20)) then
        siel(i)=cs(3)/rlp(i)
     end if
     if (th(i).gt.c(20)) then
        sieg(i)=cs(4)/rgp(i)
     end if

     !UPDATE PROPERTIES to n+1 level
     cg(i)=gas_specheat(i)
     ts(i)=gas_tempsat(i)  !ttu this section
     tg(i)=gas_temp(i); tl(i)=liq_temp(i)
     rg(i)=gas_density(i)
     if (rgp(i).le.0.0) then
        rgp(i)=0.0; rgpz(i)=0.0; rh2p(i)=0.0; rncgp(i)=0.0
     else
        xrncgp=rncgp(i)/rgp(i)
        xrh2p=rh2p(i)/rgp(i)
        xrgpz=1-xrncgp-xrh2p
        if (xrgpz.lt.0.0) xrgpz=0.0
        rgp(i)=rg(i)*th(i)
        rncgp(i)=xrncgp*rgp(i)
        rh2p(i)=xrh2p*rgp(i)
        rgpz(i)=xrgpz*rgp(i)
     end if
     rl(i)=liq_density(i)
     rlp(i)=(1.-th(i))*rl(i)
     call thbub(0)
  end do

  call siechk
  if (isiechk) then
     write(*,*)'isiechk ',isiechk
     go to 230
  end if

  !UPDATE VACUUM REGION to n+1 level if one exists
  call vacflow
  write(29,*) time,etotal

  !UPDATE PARTICLE VELOCITIES using new vapor & liquid velocities
  if (npart.eq.0) go to 161
  do i=1,ib1
     call indexf
     do k=1,npart
        if (xp(k).lt.xc(i).or.xp(k).ge.xc(ip)) go to 165
        rhomix=rgp(i)+rlp(i)
        !all the fuel particles will change thier velocities.
        !all the fuel particles are subjected to the drag forces.
        up(k)=(upn(k)+dt*(-grav*(1.-rhomix/rhop)+ &
              (dk(k)*ug(i)+ek(k)*ul(i))/pmas(k)))/ &
              (1+dt*(dk(k)+ek(k))/pmas(k))
165     continue
     end do
  end do
161 continue

  !END OF CYCLE
  do i=2,ib1
     evaptot=evaptot+dt*erate(i)*volum(i)
     fragtot=fragtot+dfrag(i)
  end do
  time=time+dt
  cycle=cycle+1

  !SET LOWER LIMIT VOID FRACTION to prevent underflow
  do i=2,ib1
     if (th(i).lt.1e-8) then
        th(i)=0.0
        rgp(i)=0.0; rh2p(i)=0.0; rgpz(i)=0.0; rncgp(i)=0.0
        cg(i)=gas_specheat(i) !ttu added
        ts(i)=gas_tempsat(i) !ttu added
        tg(i)=ts(i)
        rg(i)=gas_density(i)
        siegs(i)=gas_internalenergysat(i)
        sieg(i)=gas_internalenergy(i)
        sieg(i)=gas_internalenergyreset(i)
     end if
  end do

  go to 170
190 continue
  stop
end program main

subroutine newfrag
  include 'fcicom.for'

  integer kfront(ngmax)

  !Declare temporary variables.
  front=0.0
  wecrip=c(31)
  romix=0.0
  relvel=0.0
  dtplus=0.0
  dratio=0.0
  weber=0.0
  ab = 0.0
  rpnew=0.0
  vnew=0.0
  newp=0
  ntest=0
  nsplit=0
  isplit=0
  ntemp=0
  ntemp1=0
  j=0
  brkrate=0.0
  drprad=0.0
  drprad2=0.0
  delta=0.0
  xkmin=0.0
  xnmax=0.0
  visratio=0.0
  hbottom=0.0
  xpk=0.0
  upk=0.0
  rpartk=0.0
```

```
      upkfront=0.0
      numpj=0
      rpartj=0.0
      tpj=0.0
      totmassk=0.0
      cf0=c(141)
      thcr=c(142)
      maxn=int(c(143))
      sn1 = 0.0
      sn2 = 0.0

      !'kfront' is used to model the BL front and feeding end of BL front.
      !Currently the model assumes two leading injected partciles as the
      !BL front (one leading, one feeding).  Change 'nfront' if more
      !number of feeding particles is desired, for the time being, 'nfront'
      !is limited at 'ngmax.'  Furthermore, 'nactual' is used to count the
      !actual number of the leading front being traced.  Do not be confused
      !as 'nfront' is the limit given for BL front and, thus, is the maximum
      !of 'nactual.'  With 'nactual' as the number of BL front being traced,
      !only nactual-1 of BL front particles participate in stripping process.
      do j=1, nfront
         kfront(j)=0
      end do

      !Start the instruction.
      if (c(30).eq.0.0) return
      if (iexplos.eq.1) hbottom=dxtrig
      if (ientry.ne.2) then
         call frag
      else
         !The fragmentation process according to the new model is composed
         !of three processes due to RT and KH instabilities and BL stripping.
         !It is postulated that BL is the major cause of the breakup process
         !occur at the leading front of the injected fuel jet.  On the other
         !hand, RT is assumed responsible to the breakup process of the free
         !particles (resulted from the breaking up of the fuel jet) while as
         !KH is suggested to be the cause of the breakup of the fuel jet above
         !the jet front).  To apply this scheme on the fragmentation
         !process, the particles are divided in to three zones.  These zones
         !are leading front of the jet, above the leading front and free
         !particles.
         !In considering the free particles, parameter "ifrag" is to be
         !checked.  If it is "1", the particles are considered free. Otherwise,
         !they are considered intact and forming the jet column.
         do k=1, npartn
            nfrag(k)=0
            if (xp(k).le.hbottom) goto 1000
            if (ifrag(k).eq.1.and.tp(k).gt.tmelt) then
               !RT induced fragmentation
               i=1
2000           i=i+1
               if (i.gt.ib1) goto 2010
               if (xp(k).lt.xb(i-1).or.xp(k).gt.xb(i)) goto 2000
2010           if (i.le.ib1) then
               call stprprty(weber,dratio,dtplus,relvel,romix)
                  if (wen(k).lt.wecrip) wen(k)=weber
                  if (weber.ge.wecrip) then
                     if (c(50).ne.0.0) then
                        ab=c(49)+c(50)*dratio**0.5
                     else
                        ab=c(49)*(1.0-c(52)*(1.0-wen(k)/weber))
                        wen(k)=weber
                     endif
                     rpnew=rpold(k)*(1.0-ab*dtplus**c(51)*weber**c(52))
                     vnew=pi*rpnew**3/0.75
                     oldms=pmas(k)*nump(k)
                     newp=pmas(k)/(vnew*rhop)
                     ntest=1
                     nsplit=nump(k)
                     if (newp.gt.ntest) then
                        nump(k)=newp*nsplit
                        vnew=oldms/(rhop*nump(k))
                        rpart(k)=(vnew*0.75/pi)**(1.0/3.0)
```

```
                 pmas(k)=vnew*rhop
                 rpold(k)=rpart(k)
                 timedt=time+0.1*dt
                 if (timedt.ge.tgroup(k)) then
                    isplit=1
                    tgroup(k)=tgroup(k)+tgt
                 else
                    isplit=0
                 endif
              else
                 isplit=0
                 rpold(k)=rpnew
              endif
              if (npart.lt.npmax.and.iexplos.eq.0) then
                 if ((npart.lt.ntotal.and.isplit.ne.0).or. &
                      nump(k).ge.maxn) then
                    ntemp=0.5*nump(k)
                    if (ntemp.le.0) ntemp=1
                    ntemp1=nump(k)
                    if (ntemp*2.lt.nump(k)) then
                       nump(k)=ntemp+1
                    else
                       nump(k)=ntemp
                    endif
                    npart=npart+1
                    j=npart
                    call dupk2j(j,k)
                    nfrag(k)=j
                    !nump(j)=ntemp1-ntemp
                                   nump(j)=ntemp1-nump(k)      !Urith
                 endif
              endif
           endif
        endif
     endif
1000 continue
  end do

  !Check for the leading front of the jet.
  nactual=0
  do k=1, npartn
     if (xp(k).le.hbottom) goto 1050
     if (nactual.lt.nfront) then
        if (tp(k).gt.tmelt.and.ifrag(k).eq.0) then
           nactual=nactual+1
           i=nactual
1060       i=i-1
           if (i.eq.0) goto 1070
           if (xp(k).ge.xp(kfront(i))) goto 1070
           kfront(i+1)=kfront(i)
           goto 1060
1070       kfront(i+1)=k
        endif
     endif
1050 continue
  end do

  !BL striping
  k=kfront(1)
  if ((k.ne.0).and.(nactual.lt.nfront)) then
     do i=1, nactual
        tgroup(kfront(i))=time
     end do
  endif
  if ((k.eq.0).or.(nactual.lt.nfront)) goto 7000

  !Calculate the position, radius and falling velocity of the
  !fictitious front.
  !This fictitious front is calculated as being a column of fuel formed
  !ny 'nactual' number of leading particles.  The radius of this column
  !is calculated so that the total mass is conserved and that the length
  !of the column is that between the point on the leading surface of the
  !lowest particle and the point on the trailing surface of the toppest
```

```
     !particle.  The velocity is calculated so that the momentum is
     !conserved.
     xpk=xp(k)-rpart(k)
     rpartk=0.0
     upkfront=0.0
     totmassk=0.0
     do i=1, nactual
        totmassk=totmassk+pmas(kfront(i))*nump(kfront(i))
        upkfront=upkfront+pmas(kfront(i))*nump(kfront(i))* &
             up(kfront(i))
        rpartk=rpartk+rpart(kfront(i))**3
     end do
     upkfront=upkfront/totmassk
     rpartk=(rpartk/0.75/ &
          (xp(kfront(nactual))+rpart(kfront(nactual))-xpk))**0.5

     !With the calculated front particle, the breaking rate is calculated
     i=1
3000 i=i+1
     if (i.gt.ib1) goto 3010
     if (xpk.lt.xb(i-1).or.xpk.gt.xb(i)) goto 3000
3010 if (i.le.ib1) then
        upk=up(k)
        up(k)=upkfront
        call stprprty(weber,dratio,dtplus,relvel,romix)
        up(k)=upk
        drprad=c(32)*wecrip/2.0/romix/relvel**2.0

        !The size of the being formed fuel fragments was assumed to be
        !that governed by critical Weber number.
        visratio=1.0/ &
             (1.0+c(4)*(rgp(i)+rlp(i))/(rgp(i)*c(25)+rlp(i)*c(23)))
        delta=(c(4)*pi*rpartk*2.0/relvel/visratio/rhop)**0.5

        drprad2=delta/2.0
        if (drprad.lt.drprad2) drprad=drprad2
        !If the radius of the striped jet is comparable to that of the
        !being formed droplets, the striping process becomes less
        !effective compared to the RT process.  Here, the criteria is
        !set that BL is effective until the radius of the droplet is as
        !large as the radius of the jet.
        !Stripping rate is
        !brkrate ~ rhop*Integrate[2 Pi Ui Exp[-r/delta] R,r,{0,delta}]
        !        ~ 3.9717 rhop R Ui delta
        brkrate=3.9717*rhop*rpartk &
             *relvel*visratio*delta*(time-tgroup(k))
        vnew=pi*drprad**3/0.75

        !Once the fragmentation rate times the time difference between
        !the current time and the time it was last stripped is greater
        !then the actual mass of the first leading particle is assumed
        !totally stripped.  On the other hand, if the critical radius is
        !calculated to be greater than that of the particle, the RT
        !process is assumed to dominate the fragmentation process.  In
        !any cases, the leading particle is totally broken up.
        if (brkrate.ge.pmas(k)) then
           ifrag(k)=1
           if (kfront(2).ne.0) tgroup(kfront(2))=time
           tgroup(k)=time
           ntotal=ntotal+ngroup
        else

           !In case the leading particle is intact, it is to be checked
           !whether the stripped mass is enough to form the new
           !particles.  If it is, further investigation is if this is
           !its first stripping.  If it is then the new master group is
           !created.  Otherwise, the newly fomred particles are added
           !to the existing group.
           if (brkrate.lt.vnew*rhop) then
              !Since the mass of the fuel that has been broken up is
              !not enough to form a new particle, the broken up fuel
              !must be tracked.  Since the breakup rate is varied with
              !time, without a new variable to track the breakup rate,
            !the following line is written such that the actual
            !breakup may be estimated with the newly calculated
            !breakup rate.
            tgroup(k)=tgroup(k)+dt*(1.0-brkrate/vnew/rhop)/2.0
         else
            tgroup(k)=time
            pmas(k)=pmas(k)-brkrate
            rpart(k)=(pmas(k)/rhop*0.75/pi)**(1.0/3.0)
            if ((idghtr(k).eq.0).or.(idghtr(k).ne.0 &
                .and.nump(idghtr(k)).ge.maxn)) then
               npart=npart+1
               j=npart
               call dupk2j(j,k)
               idghtr(k)=j
               ifrag(j)=1
               nump(j)=brkrate/(vnew*rhop)*nump(k)
               pmas(j)=brkrate*nump(k)/nump(j)
               rpart(j)=(pmas(j)/rhop*0.75/pi)**(1.0/3.0)
               ntotal=ntotal+ngroup
            else
               j=idghtr(k)
               oldms=pmas(j)*nump(j)+brkrate*nump(k)
               numpj=brkrate/(vnew*rhop)*nump(k)
               rpartj=(brkrate*nump(k)/numpj/rhop*0.75/pi) &
                    **(1.0/3.0)
               tpj=(pmas(j)*nump(j)*tp(j)+brkrate*tp(k) &
                    *nump(k))/oldms
               up(j)=(pmas(j)*nump(j)*up(j)+brkrate*up(k) &
                    *nump(k))/oldms
               xp(j)=(pmas(j)*nump(j)*xp(j)*tp(j) &
                    +brkrate*xp(k)*nump(k)*tp(k))/oldms/tpj
               rpart(j)=((numpj*rpartj**2*(tp(k)-ts(i))+ &
                    nump(j)*rpart(j)**2*(tp(j)-ts(i)))/ &
                    ((numpj+nump(j))*(tpj-ts(i))))**0.5
               nump(j)=oldms/(rhop*pi*rpart(j)**3/0.75)
               pmas(j)=oldms/nump(j)
               rpart(j)=(pmas(j)/rhop*0.75/pi)**(1.0/3.0)
               rpold(j)=rpart(j)
               tp(j)=tpj
            endif
         endif
      endif
   endif
   !KH induced fragmentation
   do k=1, npartn
      if (xp(k).ge.xp(kfront(1)).and. &
          ifrag(k).eq.0.and.tp(k).gt.tmelt) then
         i=1
4000     i=i+1
         if (i.gt.ib1) goto 4010
         if (xp(k).lt.xb(i-1).or.xp(k).gt.xb(i)) goto 4000
4010     if (i.le.ib1) then
            call stprprty(weber,dratio,dtplus,relvel,romix)
            drprad=c(32)*wecrip/2.0/romix/relvel**2.0
            !Due to the complication in treating discontinuous jet,
            !the jet is assumed to stay intact though the radii at
            !various positions along the jet may vary.
            !Using the simplified model for KH process, two different
            !sets of equations are used for thicknesses approach zero
            !and infinite.  The criteria is void fraction.  If void
            !fraction is less than a given criteria (for example,
            !thcr), the thin film is assumed, otherwise thick film
            !model is used.
            if (th(i).lt.thcr) then
               visratio=up(k)-ul(i)
               xkmin=rl(i)*rhop*visratio**2.0/ &
                    1.5/(rl(i)+rhop)/(c(24)+c(32))
               xnmax=rl(i)*rhop*xkmin**2.0*visratio**2.0/ &
                    (rl(i)+rhop)**2.0-(c(24)+c(32))* &
                    xkmin**3.0/(rl(i)+rhop)
            else
               visratio=up(k)-ug(i)
               xkmin=rg(i)*rhop*visratio**2.0/ &
```

```
             1.5/(rg(i)+rhop)/c(32)
        xnmax=rg(i)*rhop*xkmin**2.0*visratio**2.0/ &
             (rg(i)+rhop)**2.0-c(32)* &
             xkmin**3.0/(rg(i)+rhop)
     endif
     if (xnmax.lt.0) xnmax=0.0
     xnmax=xnmax**0.5
     if (xkmin.le.0.0) then
        brkrate=0.0
     else
        brkrate=rhop*8.0/3.0*pi*rpart(k)**2.0*xnmax &
             /xkmin*(time-tgroup(k))*2.0*pi*cf0
     endif
     if (xkmin.gt.0.0) then
        drprad2=pi/xkmin
        if (drprad2.lt.rpart(k)) then
           if (rpart(k)/drprad2.lt.2.0**(1.0/3.0)) then
              oldms=nump(k)*pmas(k)
              nump(k)=oldms/(rhop*pi*drprad2**3/0.75)
              pmas(k)=oldms/nump(k)
              rpart(k)=(pmas(k)/rhop*0.75/pi)**(1.0/3.0)
              rpold(k)=rpart(k)
              tgroup(k)=time
              goto 1010
           else
              if (drprad.lt.drprad2) drprad=drprad2
           endif
        else
           tgroup(k)=time
           goto 1010
        endif
     else
        tgroup(k)=time
        goto 1010
     endif

     if (c(150).eq.1.0) then
        if (rpart(k)*xkmin/pi.lt.10.0) then
           tgroup(k)=time
           goto 1010
        endif
     endif
     if (brkrate.ge.pmas(k)) then
        ifrag(k)=1
        tgroup(k)=time
        ntotal=ntotal+ngroup
     else
        vnew=pi*drprad**3/0.75
        if (brkrate.lt.vnew*rhop) then
           !Since the mass of the fuel that has been broken
           !up is not enough to form a new particle, the
           !broken up fuel must be tracked.  Since the breakup
           !rate is varied with time, without a new variable
           !to track the breakup rate, the following line is
           !written such that the actual breakup may be
           !estimated with the newly calculated breakup rate.
           tgroup(k)=tgroup(k)+dt* &
                (1.0-brkrate/vnew/rhop)/2.0
        else
           tgroup(k)=time
           pmas(k)=pmas(k)-brkrate
           rpart(k)=(pmas(k)/rhop*0.75/pi)**(1.0/3.0)
           rpold(k)=rpart(k)
           if ((idghtr(k).eq.0).or.(idghtr(k).ne.0 &
                .and.nump(idghtr(k)).ge.maxn)) then
              npart=npart+1
              j=npart
              call dupk2j(j,k)
              idghtr(k)=j
              ifrag(j)=1
              nump(j)=brkrate/(vnew*rhop)*nump(k)
              pmas(j)=brkrate*nump(k)/nump(j)
              rpart(j)=(pmas(j)/rhop*0.75/pi)**(1.0/3.0)
```

```
                       rpold(j)=rpart(j)
                    else
                       j=idghtr(k)
                       oldms=pmas(j)*nump(j)+brkrate*nump(k)
                       numpj=brkrate/(vnew*rhop)*nump(k)
                       if (numpj.le.0) numpj=1
                       rpartj=(brkrate*nump(k)/numpj/rhop*0.75/pi) &
                            **(1.0/3.0)
                       tpj=(pmas(j)*nump(j)*tp(j)+brkrate*tp(k) &
                            *nump(k))/oldms
                       up(j)=(pmas(j)*nump(j)*up(j)+brkrate*up(k) &
                            *nump(k))/oldms
                       xp(j)=(pmas(j)*nump(j)*xp(j)*tp(j) &
                            +brkrate*xp(k)*nump(k)*tp(k))/ &
                            oldms/tpj
                       drprad=(numpj*rpartj**2*(tp(k)-ts(i))+ &
                            nump(j)*rpart(j)**2*(tp(j)-ts(i)))/ &
                            ((numpj+nump(j))*(tpj-ts(i)))
                       if (drprad.gt.0.0) then
                          rpart(j)=drprad**0.5
                       else
                          rpart(j)=((numpj*rpartj**2+nump(j) &
                               *rpart(j)**2)/(numpj+nump(j)))**0.5
                       endif
                       nump(j)=oldms/(rhop*pi*rpart(j)**3/0.75)
                       pmas(j)=oldms/nump(j)
                       rpart(j)=(pmas(j)/rhop*0.75/pi)**(1.0/3.0)
                       rpold(j)=rpart(j)
                       tp(j)=tpj
                    endif
                 endif
              endif
           endif
        endif
1010    continue
     end do
7000 continue
     do k=1,npart
        if (ifrag(k).ne.0.and.k.ne.kfront(1)) then
           tgroup(k)=tgroup(k)+2.0*tgt
        endif
     end do
  endif

end subroutine newfrag

subroutine dks
  include 'fcicom.for'

  ar(v1,v2)=(dx(ip)*v1+dx(i)*v2)/dxr
  dxr=dx(i)+dx(ip)
  dxl=dx(i)+dx(im)

  !calculate the homogeous mixture properties
  rgpa=ar(rgp(ih),rgp(i))
  rlpa=ar(rlp(ih),rlp(i))
  thpa=ar(thp(ih),thp(i))
  amug=ar(mug(ih)*th(ih),mug(i)*th(i))
  amul=ar(mul(ih)*(1-th(ih)),mul(i)*(1-th(i)))

  !mass-averaged mixture velocity
  !bubbly flow, transition flow regime or droplet flow regime

  !if (xp(k).lt.xc(ih).and.xp(k).ge.xb(ih)) then    !Urith
  !   if (imap(ih).eq.1) then
  !      rlpa2=rlpa
  !      romix=rgpa+rlpa
  !      vmix=(rgpa*ug(i)+rlpa2*ul(i))/romix
  !      amumix=amug+amul
  !   else if (imap(ih).eq.3) then
  !      romix=rgpa
  !      vmix=ug(i)
  !      amumix=amug
  !      rlpa2=0.
  !   else if (imap(ih).eq.2) then
  !      rlpa2=rlpa*(1-th(ih)-thlg(ih))/(1-th(ih))
  !      romix=rgpa+rlpa2
  !      vmix=(rgpa*ug(i)+rlpa2*ul(i))/romix
  !      amumix=amug+rlpa2*amul/rlpa
  !   end if

  !else if (xp(k).lt.xb(ip).and.xp(k).ge.xc(i)) then

  if (xp(k).lt.xb(ip).and.xp(k).ge.xc(i)) then
     if (imap(i).eq.1) then
        rlpa2=rlpa
        romix=rgpa+rlpa
        vmix=(rgpa*ug(i)+rlpa*ul(i))/romix
        amumix=amug+amul
     else if (imap(i).eq.3) then
        romix=rgpa
        vmix=ug(i)
        amumix=amug
        rlpa2=0.
     else if (imap(i).eq.2) then
        rlpa2=rlpa*(1-th(i)-thlg(i))/(1-th(i))
        romix=rgpa+rlpa2
        vmix=(rgpa*ug(i)+rlpa2*ul(i))/romix
        amumix=amug+rlpa2*amul/rlpa
     end if
  end if

  !relative velocity between mixture velocity and kth fuel
  !pareticles velocity, reynold number is calculated by
  !flow pass through a single spherical particle.
  relvel=abs(vmix-up(k))
  reyold=relvel*2.*rpart(k)*romix/amumix
  reyold=amax1(reyold,1e-4)

  !from the single particle reyold value decides the
  !viscous regime or newtonian regime calculating the drag
  !force of the flow on the single particle
  if (c(30).eq.0) then
     cd1=6.*(rpart(k)/rpold(k))
     cd2=0.44*(rpart(k)/rpold(k))
  else
     cd1=10.568*(rpart(k)/rpold(k))
     cd2=0.775*(rpart(k)/rpold(k))
  end if

  if (reyold.le.1000) then
     demix=cd1*pi*rpart(k)*amumix*(1.+0.15*reyold**0.687)
  else
     demix=0.5*romix*cd2*relvel*pi*rpart(k)**2
  end if

  !multiple fuel particles's drag force
  demix=demix/(1.-thpa)**2.7

  !gas-particle drag
  dk(k)=rgpa*demix/romix
  !liquid-particle drag
  ek(k)=rlpa2*demix/romix

  div=1.+dt*(ek(k)+dk(k))/pmas(k)
  sg(i)=sg(i)+dk(k)*nump(k)/div
  rgd(i)=rgd(i)+dk(k)*nump(k)*upn(k)/div
  add=ek(k)*nump(k)/div
  sl(i)=sl(i)+add
  rld(i)=rld(i)+add*upn(k)
  xu(i)=xu(i)+nump(k)*ek(k)*dk(k)/div/pmas(k)

end subroutine dks

subroutine echk
  include 'fcicom.for'
  !calculate the energies of the coolant and fuel for
  !energy conservation checking.

  !calculate the fuel energy
  if (iexplos.eq.0) ntemp=npart
  if (iexplos.eq.1) ntemp=2*npart
  efuelp=0.
  if (npart.ne.0) then
     do k=1,ntemp
        efuelp=efuelp+nump(k)*pmas(k)*siep(k)
     end do
  endif

  !calculate the coolant energies
  eliq=0.
  egas=0.
  do i=2,ib1
     egas=egas+rgp(i)*(sieg(i)+0.5*ug(i)**2.0)*volum(i)*(1.-thp(i))
     eliq=eliq+rlp(i)*(siel(i)+0.5*ul(i)**2.0)*volum(i)*(1.-thp(i))
  end do

  !calculate the energies out of the system
  dew=0.
  do i=2,ib1
         !dew=qlw(i)*(tl(i)-tw(i))*volum(i)*dt &
     dew=dew+qlw(i)*(tl(i)-tw(i))*volum(i)*dt &        !Urith
          +qgw(i)*(tg(i)-tw(i))*volum(i)*dt
  end do

  rig=rgp(ib1)*sieg(ib1)
  rigp=rgp(ib2)*sieg(ib2)
  dfeg=flxr(ug(ib1),rig,rigp)*areaj(ib1)/(1.-thp(ib1))*dt

  ril=rlp(ib1)*siel(ib1)
  rilp=rlp(ib2)*siel(ib2)
  dfel=flxr(ul(ib1),ril,rilp)*areaj(ib1)/(1.-thp(ib1))*dt

  eout=eout+(deout+dfeg+dfel+dew)

  !the total energy of both inside and outside the system
  if (c(98).eq.0.) then
     etotal=efuelp+eliq+egas+eout
  else
     etotal=eliq+egas+eout
  endif

  !calculate the energy differences
  defuel=efuelo-efuelp
  degas=egas-egaso
  deliq=eliq-eliqo
  detotal=etotal-etotalo

end subroutine echk
```

References

[B.1] Roland Span, Eric W. Lemmon, Richard T Jacobsen, Wolfgang Wagner, and Akimichi Yokozeki. A reference equation of state for the thermodynamic properties of nitrogen for temperatures from 63.151 to 1000 K and pressures to 2200 MPa. J.Phys.Chem.Ref.Data. Vol.29, No.6, 2000.

APPENDIX C

SAMPLES OF INPUT FILES FOR SIMULATION

C.1 Water at 1 CC, 77K Drop into 77K Liquid Nitrogen with 77K Wall Temperature

```
0
WATER 1CC 77K ,LN2 77K, SKIP QWALL, TW 77K

&ISET
IB=25, IPOPT=0, FLB=2, FLT=2, ITMAX=5,
THSTAR=0.5,EPSL=1E-5, EPSG=3E-3, EPSD=0.5, EPSI=0.000005,EPSP=0.5,
THFLAG=.01,ETH=.1 &

&GRID
XPMAX = .305,
DXI(1) =.04,            NDX(1)=25,
ARIY(1)=.0080914,       NARIY(1)=25,
ARJ(1) =.0080914,       NARJ(1)=25, &

&INIT
UGO=0.,         ULO=0.,
PO(1)=1E5,      NPO(1)=25,
THO(1)=0.,      NTH(1)=6,
THO(2)=1.,      NTH(2)=19,
TGO(1)=77.244, TLO(1)=77.244, NTG(1)=15,
TGO(2)=77.244, TLO(2)=77.244, NTG(2)=10,
GRAVO(1)=9.8,   NGRAV(1)=25,
TWO=77.244 &

&PART
TTLMSS=0.001,
NBREAK=1, .
RPARN=.0075, KFUEL=2.1, NBOTTOM=1,
MPPART=1, NPIN=1, CP=4.179e3,CP_ICE=2.09E3, RHOP=0.9980e3, PHEAT=3.33D5,
TMELT=273.16,
TPIN=77.244, NPARN=0, NGROUP=2, UPIN=-5.0, IENTRY=2, IENTRY2=1,
RGRP(1)=0.0075, FRCTGRP(1)=0.095, UPINGRP(1)=-3.0, TPINGRP(1)=77.244 &

&BOUND
PIN=1.0D6, THOUT=1.0, POUT=1D6 &

&RUNTIM
TPT=.0278, TIJEND=0.001, TGT=5D-2, TMAX=2.5,
DT=1D-4, DTMAX=1D-4, DTMIN=1d-7 &

&OUTPUT
LPR=3, TPL=0.1, TPL99=.1, TPL15=.1,
IPR(1)=9,IPR(2)=9,IPR(3)=9,IPR(4)=9,
IPR(5)=26,IPR(6)=26,IPR(7)=26,IPR(8)=26 &

&EXPLO

 &

&CONST
C(4)=1.75E-3,
C(5)=77.244, C(9)=806.6, C(11)=1E5,
C(13)=7.182E-3, C(14)=0.13915, C(23)=1.579E-4,C(24)=6.2e-3, C(25)=0.0553E-4,
C(20)=0.5, C(21)=1.,C(29)=7.5, C(30)=1., C(31)=12., C(32)=0.0757, C(33)=0., C(34)=0.,
C(35)=1., C(36)=1., C(37)=0., C(38)=1., C(39)=1., C(40)=1.,
C(41)=0.1,C(42)=1D-4,C(43)=0.,C(44)=0.,C(45)=0.1,C(46)=0.,C(47)=1D-3,
C(48)=1., C(49)=0.1093, C(50)=-0.0785, C(51)=1.0, C(52)=0.246,
C(53)=0., C(54)=1., C(55)=0., C(56)=1.0, C(57)=0., C(58)=0.,
C(59)=0.1,C(60)=0.,
C(61)=0.005,
C(62)=0.0,     C(63)=2.5,     C(64)=0.0,          C(65)=1.0,
C(66)=1,       C(67)=0,       C(68)=0.1,          C(69)=1.1,
C(70)=3,       C(71)=0,       C(72)=-197.0,       C(73)=-192.0,
C(74)=4,       C(75)=0,       C(76)=-197.0,       C(77)=-187.0,
C(78)=2,       C(79)=0,       C(80)=0.08,         C(81)=0.18,
C(82)=1,       C(83)=0,       C(84)=0.6,          C(85)=1.1,
C(86)=3,       C(87)=0,       C(88)=-197.0,       C(89)=-192.0,
C(90)=4,       C(91)=0,       C(92)=-197.0,       C(93)=-187.0,
C(94)=2,       C(95)=0,       C(96)=0.08,         C(97)=0.18,
C(98)=0.,C(100)=0.0,C(101) = 0.0,C(102) = 0.05,C(103) = 3.083E8,C(104) = 151.0,
C(105)=91.22,C(106) = 2.0,C(107) = 1,C(119) = 7.3D6, C(120) = 9.9D6,
C(131) = 0.0000134, C(132) = 0.285, C(133) = 14600.,C(134) = 5710000., C(135) = 400.0,
C(136) = 0.0,C(141) = 0.01,C(143) = 1.0E6, C(160)=0.0 &
```

C.2 Water at 1 CC, 77K Drop into 77K Liquid Nitrogen with Wall Temperature at 273K

```
0
WATER 1CC 77K ,LN2 77K, USE QWALL, TW 273K

&ISET
IB=25, IPOPT=0, FLB=2, FLT=2, ITMAX=5,
THSTAR=0.5,EPSL=1E-5, EPSG=3E-3, EPSD=0.5, EPSI=0.000005,EPSP=0.5,
THFLAG=.01,ETH=.1 &

&GRID
XPMAX = .305,
DXI(1) =.04,            NDX(1)=25,
ARIY(1)=.0080914,       NARIY(1)=25,
ARJ(1) =.0080914,       NARJ(1)=25, &


&INIT
UGO=0.,         ULO=0.,
PO(1)=1E5,      NPO(1)=25,
THO(1)=0.,      NTH(1)=6,
THO(2)=1.,      NTH(2)=19,
TGO(1)=77.244, TLO(1)=77.244, NTG(1)=15,
TGO(2)=77.244, TLO(2)=77.244, NTG(2)=10,
GRAVO(1)=9.8,  NGRAV(1)=25,
TWO=273 &

&PART
TTLMSS=0.001,
NBREAK=1,
RPARN=.0075, KFUEL=2.1, NBOTTOM=1,
MPPART=1, NPIN=1, CP=4.179e3,CP_ICE=2.09E3, RHOP=0.9980e3, PHEAT=3.33D5,
TMELT=273.16,
TPIN=77.244, NPARN=0, NGROUP=2, UPIN=-5.0, IENTRY=2, IENTRY2=1,
RGRP(1)=0.0075, FRCTGRP(1)=0.095, UPINGRP(1)=-3.0, TPINGRP(1)=77.244 &

&BOUND
PIN=1.0D6, THOUT=1.0, POUT=1D6 &

&RUNTIM
TPT=.0278, TIJEND=0.001, TGT=5D-2, TMAX=2.5,
DT=1D-4, DTMAX=1D-4, DTMIN=1d-7 &

&OUTPUT
LPR=3, TPL=0.1, TPL99=.1, TPL15=.1,
IPR(1)=9,IPR(2)=9,IPR(3)=9,IPR(4)=9,
IPR(5)=26,IPR(6)=26,IPR(7)=26,IPR(8)=26 &

&EXPLO

 &

&CONST
C(4)=1.75E-3,
C(5)=77.244, C(9)=806.6, C(11)=1E5,
C(13)=7.182E-3, C(14)=0.13915, C(23)=1.579E-4,C(24)=6.2e-3, C(25)=0.0553E-4,
C(20)=0.5, C(21)=1.,C(29)=7.5, C(30)=1., C(31)=12., C(32)=0.0757, C(33)=0., C(34)=0.,
C(35)=1., C(36)=1., C(37)=0., C(38)=1., C(39)=1., C(40)=1.,
C(41)=0.1,C(42)=1D-4,C(43)=0.,C(44)=0.,C(45)=0.1,C(46)=0.,C(47)=1D-3,
C(48)=1., C(49)=0.1093, C(50)=-0.0785, C(51)=1.0, C(52)=0.246,
C(53)=0., C(54)=1., C(55)=0., C(56)=1.0, C(57)=0., C(58)=0.,
C(59)=0.1,C(60)=0.,
C(61)=0.005,
C(62)=0.0,     C(63)=2.5,     C(64)=0.0,           C(65)=1.0,
C(66)=1,       C(67)=0,       C(68)=0.1,           C(69)=1.1,
C(70)=3,       C(71)=0,       C(72)=-197.0,        C(73)=-192.0,
C(74)=4,       C(75)=0,       C(76)=-197.0,        C(77)=-187.0,
C(78)=2,       C(79)=0,       C(80)=0.08,          C(81)=0.18,
C(82)=1,       C(83)=0,       C(84)=0.6,           C(85)=1.1,
C(86)=3,       C(87)=0,       C(88)=-197.0,        C(89)=-192.0,
C(90)=4,       C(91)=0,       C(92)=-197.0,        C(93)=-187.0,
C(94)=2,       C(95)=0,       C(96)=0.08,          C(97)=0.18,
C(98)=0.,C(100)=0.0,C(101) = 0.0,C(102) = 0.05,C(103) = 3.083E8,C(104) = 151.0,
C(105)=91.22,C(106) = 2.0,C(107) = 1,C(119) = 7.3D6, C(120) = 9.9D6,
C(131) = 0.0000134, C(132) = 0.285, C(133) = 14600.,C(134) = 5710000., C(135) = 400.0,
C(136) = 0.0,C(141) = 0.01,C(143) = 1.0E6, C(160)=1.0 &
```

C.3 Water Injection 2-bar and Volumetric Ratio 0.05

```
0
WATER 2bar 100CC 300K ,LN2 77K, USE QWALL, TW 273K

&ISET
IB=25, IPOPT=0, FLB=2, FLT=2, ITMAX=5,
THSTAR=0.5,EPSL=1E-5, EPSG=3E-3, EPSD=0.5, EPSI=0.000005,EPSP=0.5,
THFLAG=.01,ETH=.1 &

&GRID
XPMAX = .305,
DXI(1) =.04,           NDX(1)=25,
ARIY(1)=.0080914,      NARIY(1)=25,
ARJ(1) =.0080914,      NARJ(1)=25, &


&INIT
UGO=0.,         ULO=0.,
PO(1)=1E5,      NPO(1)=25,
THO(1)=0.,      NTH(1)=6,
THO(2)=1.,      NTH(2)=19,
TGO(1)=77.244, TLO(1)=77.244, NTG(1)=15,
TGO(2)=77.244, TLO(2)=77.244, NTG(2)=10,
GRAVO(1)=9.8,   NGRAV(1)=25,
TWO=273 &

&PART
TTLMSS=0.100,
NBREAK=1,
RPARN=.0075, KFUEL=2.1, NBOTTOM=1,
MPPART=1, NPIN=1, CP=4.179e3,CP_ICE=2.09E3, RHOP=0.9980e3, PHEAT=3.33D5,
TMELT=273.16,
TPIN=300, NPARN=0, NGROUP=2, UPIN=-5.0, IENTRY=2, IENTRY2=1,
RGRP(1)=0.0075, FRCTGRP(1)=0.095, UPINGRP(1)=-0.88, TPINGRP(1)=300 &

&BOUND
PIN=1.0D6, THOUT=1.0, POUT=1D6 &

&RUNTIM
TPT=.0278, TIJEND=0.001, TGT=5D-2, TMAX=2.5,
DT=1D-4, DTMAX=1D-4, DTMIN=1d-7 &

&OUTPUT
LPR=3, TPL=0.1, TPL99=.1, TPL15=.1,
IPR(1)=9,IPR(2)=9,IPR(3)=9,IPR(4)=9,
IPR(5)=26,IPR(6)=26,IPR(7)=26,IPR(8)=26 &

&EXPLO

 &

&CONST
```

```
C(4)=1.75E-3,
C(5)=77.244, C(9)=806.6, C(11)=1E5,
C(13)=7.182E-3, C(14)=0.13915, C(23)=1.579E-4,C(24)=6.2e-3, C(25)=0.0553E-4,
C(20)=0.5, C(21)=1.,C(29)=7.5, C(30)=1., C(31)=12., C(32)=0.0757, C(33)=0., C(34)=0.,
C(35)=1., C(36)=1., C(37)=0., C(38)=1., C(39)=1., C(40)=1.,
C(41)=0.1,C(42)=1D-4,C(43)=0.,C(44)=0.,C(45)=0.1,C(46)=0.,C(47)=1D-3,
C(48)=1., C(49)=0.1093, C(50)=-0.0785, C(51)=1.0, C(52)=0.246,
C(53)=0., C(54)=1., C(55)=0., C(56)=1.0, C(57)=0., C(58)=0.,
C(59)=0.1,C(60)=0.,
C(61)=0.005,
C(62)=0.0,     C(63)=2.5,     C(64)=0.0,         C(65)=1.0,
C(66)=1,       C(67)=0,       C(68)=0.1,         C(69)=1.1,
C(70)=3,       C(71)=0,       C(72)=-197.0,      C(73)=-192.0,
C(74)=4,       C(75)=0,       C(76)=-197.0,      C(77)=-187.0,
C(78)=2,       C(79)=0,       C(80)=0.1,         C(81)=0.2,
C(82)=1,       C(83)=0,       C(84)=0.6,         C(85)=1.1,
C(86)=3,       C(87)=0,       C(88)=-197.0,      C(89)=-192.0,
C(90)=4,       C(91)=0,       C(92)=-197.0,      C(93)=-187.0,
C(94)=2,       C(95)=0,       C(96)=0.1,         C(97)=0.2,
C(98)=0.,C(100)=0.0,C(101) = 0.0,C(102) = 0.05,C(103) = 3.083E8,C(104) = 151.0,
C(105)=91.22,C(106) = 2.0,C(107) = 1,C(119) = 7.3D6, C(120) = 9.9D6,
C(131) = 0.0000134, C(132) = 0.285, C(133) = 14600.,C(134) = 5710000., C(135) = 400.0,
C(136) = 0.0,C(141) = 0.01,C(143) = 1.0E6, C(160)=1.0 &
```

C.4 Water Injection 3-bar and Volumetric Ratio 0.05

```
0
WATER 3bar 100CC 300K ,LN2 77K, USE QWALL, TW 273K

&ISET
IB=25, IPOPT=0, FLB=2, FLT=2, ITMAX=5,
THSTAR=0.5,EPSL=1E-5, EPSG=3E-3, EPSD=0.5, EPSI=0.000005,EPSP=0.5,
THFLAG=.01,ETH=.1 &

&GRID
XPMAX = .305,
DXI(1) =.04,          NDX(1)=25,
ARIY(1)=.0080914,     NARIY(1)=25,
ARJ(1) =.0080914,     NARJ(1)=25, &


&INIT
UGO=0.,         ULO=0.,
PO(1)=1E5,      NPO(1)=25,
THO(1)=0.,      NTH(1)=6,
THO(2)=1.,      NTH(2)=19,
TGO(1)=77.244, TLO(1)=77.244, NTG(1)=15,
TGO(2)=77.244, TLO(2)=77.244, NTG(2)=10,
GRAVO(1)=9.8,  NGRAV(1)=25,
TWO=273 &

&PART
TTLMSS=0.100,
NBREAK=1,
RPARN=.0075, KFUEL=2.1, NBOTTOM=1,
MPPART=1, NPIN=1, CP=4.179e3,CP_ICE=2.09E3, RHOP=0.9980e3, PHEAT=3.33D5,
TMELT=273.16,
TPIN=300, NPARN=0, NGROUP=2, UPIN=-5.0, IENTRY=2, IENTRY2=1,
RGRP(1)=0.0075, FRCTGRP(1)=0.095, UPINGRP(1)=-0.89, TPINGRP(1)=300 &

&BOUND
PIN=1.0D6, THOUT=1.0, POUT=1D6 &

&RUNTIM
TPT=.0278, TIJEND=0.001, TGT=5D-2, TMAX=2.5,
DT=1D-4, DTMAX=1D-4, DTMIN=1d-7 &

&OUTPUT
LPR=3, TPL=0.1, TPL99=.1, TPL15=.1,
IPR(1)=9,IPR(2)=9,IPR(3)=9,IPR(4)=9,
IPR(5)=26,IPR(6)=26,IPR(7)=26,IPR(8)=26 &

&EXPLO
```

```
 &

&CONST
C(4)=1.75E-3,
C(5)=77.244, C(9)=806.6, C(11)=1E5,
C(13)=7.182E-3, C(14)=0.13915, C(23)=1.579E-4,C(24)=6.2e-3,
C(25)=0.0553E-4,
C(20)=0.5, C(21)=1.,C(29)=7.5, C(30)=1., C(31)=12., C(32)=0.0757, C(33)=0., C(34)=0.,
C(35)=1., C(36)=1., C(37)=0., C(38)=1., C(39)=1., C(40)=1.,
C(41)=0.1,C(42)=1D-4,C(43)=0.,C(44)=0.,C(45)=0.1,C(46)=0.,C(47)=1D-3,
C(48)=1., C(49)=0.1093, C(50)=-0.0785, C(51)=1.0, C(52)=0.246,
C(53)=0., C(54)=1., C(55)=0., C(56)=1.0, C(57)=0., C(58)=0.,
C(59)=0.1,C(60)=0.,
C(61)=0.005,
C(62)=0.0,     C(63)=2.5,     C(64)=0.0,           C(65)=1.0,
C(66)=1,       C(67)=0,       C(68)=0.1,           C(69)=1.1,
C(70)=3,       C(71)=0,       C(72)=-197.0,        C(73)=-192.0,
C(74)=4,       C(75)=0,       C(76)=-197.0,        C(77)=-187.0,
C(78)=2,       C(79)=0,       C(80)=0.1,           C(81)=0.2,
C(82)=1,       C(83)=0,       C(84)=0.6,           C(85)=1.1,
C(86)=3,       C(87)=0,       C(88)=-197.0,        C(89)=-192.0,
C(90)=4,       C(91)=0,       C(92)=-197.0,        C(93)=-187.0,
C(94)=2,       C(95)=0,       C(96)=0.1,           C(97)=0.2,
C(98)=0.,C(100)=0.0,C(101) = 0.0,C(102) = 0.05,C(103) = 3.083E8,C(104) = 151.0,
C(105)=91.22,C(106) = 2.0,C(107) = 1,C(119) = 7.3D6, C(120) = 9.9D6,
C(131) = 0.0000134, C(132) = 0.285, C(133) = 14600.,C(134) = 5710000., C(135) = 400.0,
C(136) = 0.0,C(141) = 0.01,C(143) = 1.0E6, C(160)=1.0 &
```

C.5 Water Injection 4-bar and Volumetric Ratio 0.10

```
0
WATER 4bar 200CC 300K ,LN2 77K, USE QWALL, TW 273K

&ISET
IB=25, IPOPT=0, FLB=2, FLT=2, ITMAX=5,
THSTAR=0.5,EPSL=1E-5, EPSG=3E-3, EPSD=0.5, EPSI=0.000005,EPSP=0.5,
THFLAG=.01,ETH=.1 &

&GRID
XPMAX = .305,
DXI(1) =.04,           NDX(1)=25,
ARIY(1)=.0080914,      NARIY(1)=25,
ARJ(1) =.0080914,      NARJ(1)=25, &


&INIT
UGO=0.,         ULO=0.,
PO(1)=1E5,      NPO(1)=25,
THO(1)=0.,      NTH(1)=6,
THO(2)=1.,      NTH(2)=19,
TGO(1)=77.244, TLO(1)=77.244, NTG(1)=15,
TGO(2)=77.244, TLO(2)=77.244, NTG(2)=10,
GRAVO(1)=9.8,   NGRAV(1)=25,
TWO=273 &

&PART
TTLMSS=0.200,
NBREAK=1,
RPARN=.0075, KFUEL=2.1, NBOTTOM=1,
MPPART=1, NPIN=1, CP=4.179e3,CP_ICE=2.09E3, RHOP=0.9980e3, PHEAT=3.33D5,
TMELT=273.16,
TPIN=300, NPARN=0, NGROUP=2, UPIN=-5.0, IENTRY=2, IENTRY2=1,
RGRP(1)=0.0075, FRCTGRP(1)=0.095, UPINGRP(1)=-2.19, TPINGRP(1)=300 &

&BOUND
PIN=1.0D6, THOUT=1.0, POUT=1D6 &

&RUNTIM
TPT=.0278, TIJEND=0.001, TGT=5D-2, TMAX=2.5,
DT=1D-4, DTMAX=1D-4, DTMIN=1d-7 &

&OUTPUT
LPR=3, TPL=0.1, TPL99=.1, TPL15=.1,
IPR(1)=9,IPR(2)=9,IPR(3)=9,IPR(4)=9,
IPR(5)=26,IPR(6)=26,IPR(7)=26,IPR(8)=26 &

&EXPLO

 &

&CONST
C(4)=1.75E-3,
C(5)=77.244, C(9)=806.6, C(11)=1E5,
C(13)=7.182E-3, C(14)=0.13915, C(23)=1.579E-4,C(24)=6.2e-3,
C(25)=0.0553E-4,
C(20)=0.5, C(21)=1.,C(29)=7.5, C(30)=1., C(31)=12., C(32)=0.0757, C(33)=0., C(34)=0.,
C(35)=1., C(36)=1., C(37)=0., C(38)=1., C(39)=1., C(40)=1.,
C(41)=0.1,C(42)=1D-4,C(43)=0.,C(44)=0.,C(45)=0.1,C(46)=0.,C(47)=1D-3,
C(48)=1., C(49)=0.1093, C(50)=-0.0785, C(51)=1.0, C(52)=0.246,
C(53)=0., C(54)=1., C(55)=0., C(56)=1.0, C(57)=0., C(58)=0.,
C(59)=0.1,C(60)=0.,
C(61)=0.005,
C(62)=0.0,     C(63)=2.5,     C(64)=0.0,          C(65)=1.0,
C(66)=1,       C(67)=0,       C(68)=0.1,          C(69)=1.1,
C(70)=3,       C(71)=0,       C(72)=-197.0,       C(73)=-192.0,
C(74)=4,       C(75)=0,       C(76)=-197.0,       C(77)=-187.0,
C(78)=2,       C(79)=0,       C(80)=0.1,          C(81)=0.2,
C(82)=1,       C(83)=0,       C(84)=0.6,          C(85)=1.1,
C(86)=3,       C(87)=0,       C(88)=-197.0,       C(89)=-192.0,
C(90)=4,       C(91)=0,       C(92)=-197.0,       C(93)=-187.0,
C(94)=2,       C(95)=0,       C(96)=0.1,          C(97)=0.2,
C(98)=0.,C(100)=0.0,C(101) = 0.0,C(102) = 0.05,C(103) = 3.083E8,C(104) = 151.0,
C(105)=91.22,C(106) = 2.0,C(107) = 1,C(119) = 7.3D6, C(120) = 9.9D6,
C(131) = 0.0000134, C(132) = 0.285, C(133) = 14600.,C(134) = 5710000., C(135) = 400.0,
C(136) = 0.0,C(141) = 0.01,C(143) = 1.0E6, C(160)=1.0 &
```

APPENDIX D

ESTIMATING THE INTERFACE TEMPERATURE OF TWO SUDDENLY CONTACTING MATERIALS

Other than the theoretical modeling in TEXAS code, Kazimi et al [D.1] also derived the interface temperature of two suddenly contacting materials. The derivation was based on the semi-infinite body of the uniform initial temperature. The interface temperature could give an estimated temperature of the contacting water and liquid nitrogen in this work. The interface temperature was expressed in terms of the temperature and the ratio of the thermal conductivity to the thermal diffusivity as:

$$T_I = \frac{T_H(k/a_t)_H + T_C(k/a_t)_C}{(k/a_t)_H + (k/a_t)_C}$$

$$= \frac{T_H(\rho c)_H + T_C(\rho c)_C}{(\rho c)_H + (\rho c)_C}$$

where T_H = Temperature of the hot material,

T_C = Temperature of the cold material,

$(k/a_t)_H$ = Ratio of the thermal conductivity to the diffusivity of the hot material,

$(k/a_t)_C$ = Ratio of the thermal conductivity to the diffusivity of the cold material,

$(\rho c)_H$ = Product of the density and the specific heat of the hot material and

$(\rho c)_C$ = Product of the density and the specific heat of the cold material.

The water with the temperature, density and specific heat of 300K, 997 kg/m^3 and 4179 J/kg-K, and the liquid nitrogen with the temperature, density and specific heat of 77K, 806 kg/m^3 and 2041 J/kg-K gave the interface temperature of 237K.

Reference

[D.1] Kazimi, M.S. and Erdman, C.A. On the Interface Temperature of Two Suddenly Contacting Materials. **J. of Heat Transfer.** (November 1975):615-617.

APPENDIX E

ESTIMATING OF THE MINIMUM FILM BOILING TEMPERATURE

Since the vapor explosion was initiated by the collapse of the film boiling, the minimum film boiling temperature was an important key for it acted as a threshold parameter during the mixing phase. Simon et al [E.1] gave an empirical expression for the minimum film boiling temperature as:

$$\frac{T_{mfb}}{T_{crit}} = 0.13\left(\frac{P}{P_{crit}}\right) + 0.86 \pm 0.06$$

where T_{mfb} = Minimum surface temperature required to support the film boiling,

T_{crit} = Critical temperature of the cold liquid,

P_{crit} = Critical pressure of the cold liquid and

P = Boiling pressure

The liquid nitrogen with its critical temperature and critical pressure of 126.192K and 3.3958 MPa, and the boiling pressure of 0.1 MPa and 0.2 MPa resulted in the minimum film boiling temperature of 109.0K±7.6K and 109.5K±7.6K, respectively. The middle temperature of 109K was less than half of the contact temperature of 237K. Thus, it was very likely that the nitrogen film boiling occurred in the experiments.

Reference

[E.6] Simon, F.F., Papell, S.S., and Simoneau, R.J. Minimum Film Boiling Heat Flux in Vertical Flow of Liquid Nitrogen. **NASA TND-4307.** 1968.

BIOGRAPHY

Mr. Urith Archakositt was born in Bangkok, Thailand on August 18, 1969. He received his Bachelor degree in electrical engineering from Chulalongkorn University in 1991 and Master degree in electrical engineering from the same university in 1994.

In 1997, he joined a Map-Ta-Phut cogeneration project as a project engineer for 3 years and as an advisor for 2 years.

In 1998, he entered Ph.D. program at Chulalongkorn University. His current interests are in the field of mathematics, fluid mechanics, heat transfer and thermal hydraulic engineering.