

LBL--31917

DE93 001533

The HILDA Program*

E. CLOSE, C. FONG, and E. LEE

Accelerator and Fusion Research Division
Lawrence Berkeley Laboratory
University of California
Berkeley, California 94720

October 30, 1991

- * This work was supported by the Director, Office of Energy Research, Office of Fusion Energy, Inertial Fusion Energy Division, U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

MASTER

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

YH

CONTENTS

1. HILDA OVERVIEW	1
2. RUNNING HILDA	3
A. Introduction	3
B. Two Simple Examples	6
Example 1: The 3MV Station of a 4MJ Driver	6
Example 2: The 3000MV Station of a 4MJ Driver	46
C. HILDA Module Data Files	65
D. Analyzing HILDA Results	66
3. THE HILDA MODEL	67
A. Introduction	67
B. Beam Transport Equations	68
C. Acceleration-Transport Elements	71
Iron Aided Pulsed Quadrupole	71
Superconducting Quadrupole	73
Acceleration Modules	78
4. MAINTAINING HILDA	87
A. Installation	87
B. Updating Conventions and Philosophy	87
C. Future Updates	88
Input	88
Output	88
Additional Costing Capabilities	88
APPENDIX: HILDA PROGRAM DOCUMENT DISKS	89
A. Guide to the HPD Disks	89
B. Installing HILDA	91

LIST OF FIGURES

Figure 2-1.	HILDA Module Data Structure	5
Figure 2-2.	HILDA Output A	11
Figure 2-3.	HILDA Output B	12
Figure 2-4.	HILDA Output C	12
Figure 2-5.	HILDA Output D	13
Figure 2-6.	HILDA Output E	17
Figure 2-7.	HILDA Output F	20
Figure 2-8.	CalCost1.DAT	27
Figure 2-9.	CostV1.DAT	28
Figure 2-10.	AlSighAbar.DAT	29
Figure 2-11.	TranMod.DAT	30
Figure 2-12.	FeQ20.DAT	30
Figure 2-13.	ScQ30.DAT	32
Figure 2-14.	StrucCore.DAT	38
Figure 2-15.	HILDA Output A	50
Figure 2-16.	HILDA Output B	50
Figure 2-17.	HILDA Output C	51
Figure 2-18.	HILDA Output D	51
Figure 2-19.	HILDA Output E	56
Figure 2-20.	HILDA Output F	58
Figure 2-21.	CalCost1.DAT	61
Figure 2-22.	CostV1.DAT	63
Figure 2-23.	AlSighAbar.DAT	64
Figure 2-24.	TranMod.DAT	64
Figure 2-25.	FeQ20.DAT	64
Figure 2-26.	ScQ30.DAT	64
Figure 2-27.	StrucCore.DAT	64
Figure 3-1.	Acceleration-Transport Module Cost	67
Figure 5-1.	HPD Floppy Disks	89
Figure 5-2.	HPD/folders: ReadME	90
Figure 5-3.	Hilda Installation	91

1. HILDA OVERVIEW

Although this report is called a program document, it is not simply a user's guide to running HILDA nor is it a programmer's guide to maintaining and updating HILDA. Instead it is a guide to HILDA as a program and as a model for designing and costing a heavy ion fusion (HIF) driver. HILDA represents the work and ideas of many people; as does the model upon which it is based. The project was initiated by Denis Keefe, who was then the leader of the LBL HIFAR project. He also suggested the name HILDA, which is an acronym for Heavy Ion Linac Driver Analysis.

The conventions and style of development of the HILDA program are based on the original goals. It was desired to have a computer program that could estimate the cost and find an optimal design for Heavy Ion Fusion induction linac drivers. This program should be able to model near-term machines as well as full-scale drivers. The code objectives were: (1) A relatively detailed, but easily understood model. (2) Modular, structured code to facilitate making changes in the model, the analysis reports, and the user interface. (3) Documentation that defines, and explains the system model, cost algorithm, program structure, and generated reports. With this tool a knowledgeable user would be able to examine an ensemble of drivers and find the driver that is minimum in cost, subject to stated constraints.

This document, referred to as the HILDA Program Document (HPD), contains a report section that describes how to use HILDA, some simple illustrative examples, and descriptions of the models used for the beam dynamics and component design. Associated with this document, as files on floppy disks, are the complete HILDA source code, much information that is needed to maintain and update HILDA, and some complete examples. These examples illustrate that the present version of HILDA can generate much useful information about the design of a HIF driver. They also serve as guides to what features would be useful to include in future updates. The HPD represents the current state of development of this project.

At its present state of development HILDA is a FORTRAN program that helps a user find minimum-cost designs of the focusing elements and the acceleration cells that are in the acceleration-transport section of a heavy ion fusion driver. A user selects stations in this section of the driver and for each station furnishes data that defines: the particle beam, the design of the focusing and acceleration components, the points in a parameter space that HILDA scans. A point in that parameter space is defined by the values of: L the lattice half-period, a the maximum beamlet envelope size, ΔV the station voltage gain, η the focusing element packing factor.

When run at a station HILDA tries to produce a design at each point in the parameter space, subject to constraints furnished in the data. If a suitable design can be produced, the design cost $\$/\Delta V$ is recorded. Upon completion of scanning all points in the parameter space, HILDA selects from the ensemble of successful designs the one that was minimum in cost and writes the complete design to a logfile.

There remains work to be done in extending the program capabilities to include the complete driver in the search for a minimum cost. The present version of HILDA does not assemble the individual station designs into a complete acceleration-transport section. However, the station logfiles that HILDA produces contain all the information that is needed to do this. Also, there remains work to be done in completing the cost model and more effort should be expended on developing the user interface. At its present stage of development all these tasks could easily be completed.

2. RUNNING HILDA

A. INTRODUCTION

HILDA has been built to find the minimum-cost design at a particular machine station. When there are many stations and the data at these stations is different, it is obvious that HILDA needs modules that provide a simple user interface for setting up the data files. The present version of HILDA does not contain these modules and thus requires an excessive amount of data handling. Future versions should correct this deficiency and thus significantly reduce the task of providing data to HILDA. Also, the creation of output files that can be read by data analysis programs should be simplified. For example, a spreadsheet program is very useful in analyzing results and it also is able to generate reports. It is now somewhat time consuming to generate input to such an analysis program using HILDA output.

The basic data structure of HILDA is shown in Figure 2-1, *Hilda Module Data Structure*. In that figure we show two stations, which are identified as IDStation = 10 and IDStation = 70. In principal there may be as many stations as the user wishes. A station is a point along the machine at which a minimum-cost design is to be produced.

The picture shows the required module data sets. These data sets, which are files that HILDA reads, have the names *module-name.DAT* and are listed below:

CalCost1.DAT	Beam parameter data
AlSighBar.DAT	Beam dynamics parameters
CostV1.DAT	Parameter minimization search space
FeQ20.DAT	Fe quadrupole design data
ScQ30.DAT	Superconducting quadrupole design data
StrucCore.DAT	Acceleration structure design data
TranMod.DAT	Transport module selection data

The beam parameter data is furnished in one data file, CalCost1.DAT, for all the stations under consideration. In that file there is a packet of data for each of the stations under consideration. This is indicated in Figure 2-1, where we have shown two stations.

The other data sets are associated with the HILDA design module of the same name. For example, the data in the file AlSighBar.DAT is used by the module AlSighBar that calculates beam-dynamic quantities. If the module's design data changes at the different stations, then the content of the data file must be updated to the correct value for the current station. The next version of HILDA will have modules that enable the user to easily set the contents of these data files.

Much of the design information that is contained in these data sets is fixed for the entire machine. For example, the price of the amorphous material does not change as a function of the machine station. However, the tape width used at a particular station might be changed. These data sets contain all the design information available to the design modules at the current station and for that reason can be rather large. Thus, we have decided to associate them with the design module rather than the machine station.

Running HILDA consists of establishing the above mentioned module data files and then invoking the HILDA main program. Once the main program starts, prompts are issued to guide the user. In the description below of how to run HILDA we shall assume that it has been installed in the user's computing environment. How to do this installation is described in the section *Maintaining Hilda* of this program document.

The beam parameter data file, CalCost1.DAT, defines the stations and the beam parameters at those stations. For each of these stations there must be a set of data files for the HILDA design modules. This set must be loaded onto the *module.DAT* files before executing HILDA. The program is then run to find the minimum-cost machine at the corresponding station. The results of the run are put on the OUTPUT files shown in Figure 2-1. The basic output file is identified as CostV1.*nnn*, where *nnn* is one of the station numbers furnished in the CalCost1.DAT file. This file contains the minimum-cost design for the station *nnn*. The logfile CostV1.LOG can be copied to the file CostV1Log.*nnn*, if it is of interest. This logfile may not be of interest; it may contain very little information. Its content depends on the data in the associated module data file.

To summarize the running of HILDA:

- define the parameters in CalCost1.DAT, the beam parameter data file
- define the data in the remaining module data files for each station in CalCost1.DAT
this data can reside on the files

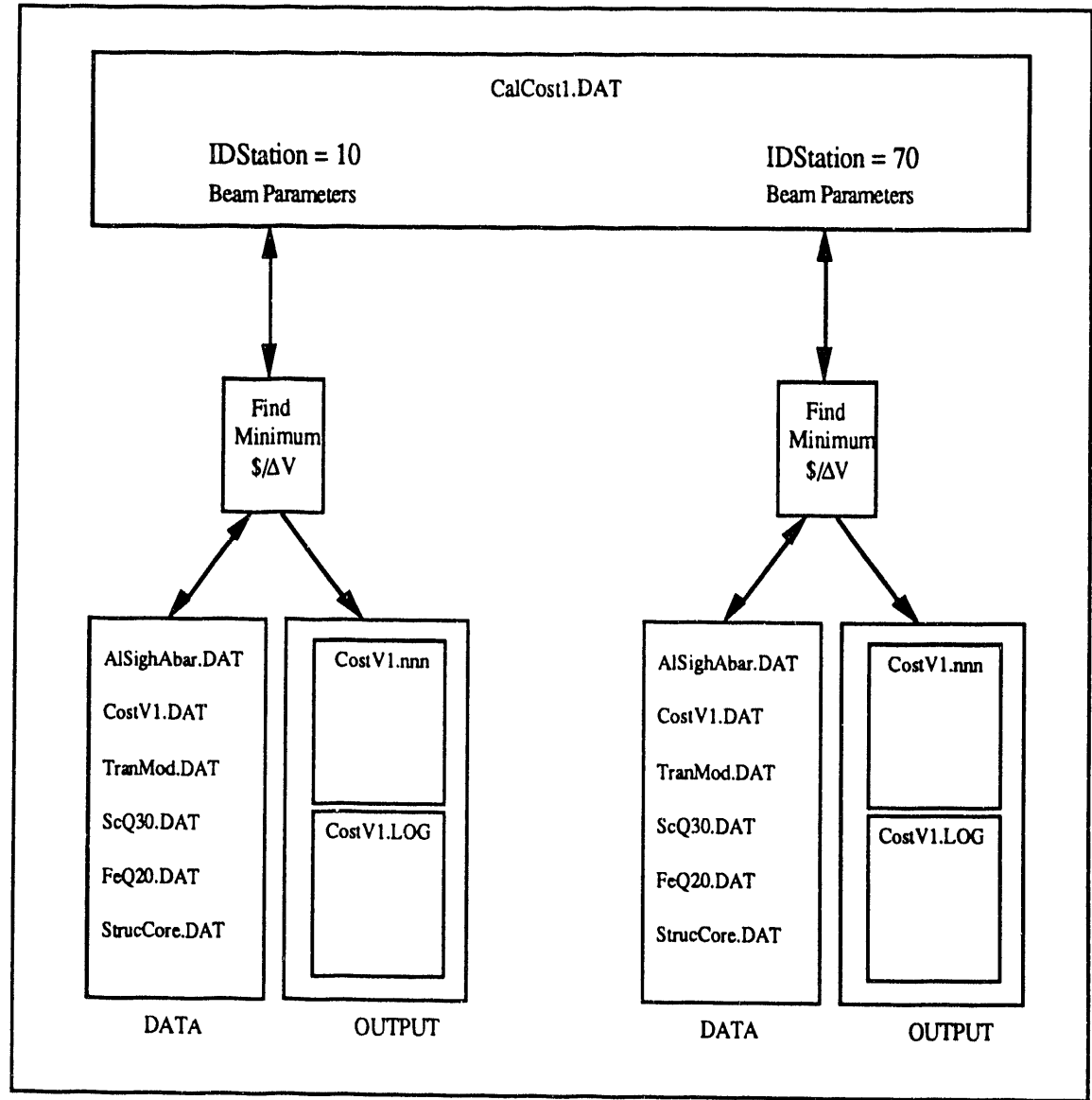
CostV1. <i>nnn</i>	Parameter search space
AlSighBar. <i>nnn</i>	Beam dynamics parameters
TranMod. <i>nnn</i>	Transport module selection data
FeQ20. <i>nnn</i>	Fe quadrupole design data
ScQ30. <i>nnn</i>	Superconducting design data
StrucCore. <i>nnn</i>	Acceleration structure data

 where *nnn* is the station number to which the data corresponds.
- transfer the data to the corresponding module data files with file type DAT
- run HILDA to find the minimum cost at the station for which the module data files are applicable
- save the file CostV1.NNN that contains the complete information for the minimum-cost design at IDStation = NNN
- analyze the output and generate reports using, for example, a spreadsheet

The initial creation of the HILDA data files is done with a word processor, or text editor. The examples furnished here can serve as templates. These files are fully formatted and their contents are meant to be self-explanatory. New data files can be created by editing the data files in the examples. We give here two simple, complete examples that serve as a guide on how to set up the data files, how to run HILDA, and how to interpret the program output obtained.

Again, we note that future editions of HILDA will not require the user to perform the file transfer needed to set up the module data files at the current station. Also, it will be possible for the user to change parameters of the data files from within the executing program. However, we emphasize here that HILDA is not a "black box" that drops out a minimum-cost machine. It has been built to interact with knowledgeable users, and thus help them arrive at an optimum machine design. As experience is gained in running HILDA its "black box" capabilities will no doubt be expanded. However, because of the complex nature of the machine design problem, the need to have a knowledgeable user will probably not change.

Figure 2-1. HILDA Module Data Structure



B. TWO SIMPLE EXAMPLES

In this section we give two complete HILDA examples. We change slightly the order of the summary outline given above in the *Introduction*. We describe first the running of HILDA. In actual order of occurrence we must first have set up the module data files for the HILDA program. However, in this example the data files already exist and we can proceed to run the program. These files contain a significant amount of data and thus require some detailed explanations. Since the parameters in these files appear in the HILDA generated output, it is necessary that a knowledgeable user understand their contents; also, these files can be used as templates when furnishing data for other HILDA runs.

Example 1: The 3MV Station of a 4MJ Driver

We point out first that the example described here is in the folder *dr4MJ @ 3MV*, which is in the folder *MSW/Hilda/DAT* of the Hilda Program Document disks. The *ReadMe* files in those folders contain additional information about this example. The information that we present here has been extracted from files in those folders. This 3MV example and all its output can be reconstructed from the data files in *dr4MJ @ 3MV*. We also note that the 3MV example set up when HILDA is installed following the instructions in the HPD section *Maintaining HILDA: Installation* is precisely the example that we are describing here

Running HILDA on the VAX

It is easy to run HILDA. In what follows we assume that:

- HILDA has been installed by following the instructions in section *Maintaining Hilda: Installation*
- you are logged onto your VAX account and that the directory is [USER.HILDA].
- you have executed the LOGIN.COM file that was loaded into this directory
- the installed executable image HILDA.EXE exists in the directory [USER.HILDA.EXE]
- the 3000MV data files were installed in the directory [USER.HILDA.DAT.DAT3000MV]

All the above assumptions will be true if you have installed HILDA as recommended.

If these assumptions are true, then follow the steps below.

- Transfer to the execution directory by typing
EXE
- Check that you are in the [USER.HILDA.EXE] directory by typing
SHOW DEFAULT
If you are not in this directory something is wrong.
- Set up the HILDA module data files by typing
SET3MV
At this point the HILDA module data files:
CalCost1.DAT
AISighBar.DAT
CostV1.DAT
TranMod.DAT
ScQ30.DAT
StrucCore.DAT
contain the 3000MV data of this example.

- Run HILDA by typing

`RUNHILDA`

The main program HILDA will execute and you will be prompted for input. Follow the prompts. We comment here on what you will see and what you can do with the present version. HILDA is a menu driven program. When you type in one of the menu commands it will proceed to do the task that you have requested. The present version of the program has a limited user interface. Future versions will have expanded capabilities. Currently we can ask HILDA to produce a minimum-cost design at a selected station and we can request that it terminate execution; i.e., stop and return control to the operating environment.

- Continue with the example by entering in lower case the command

`cost`

Notice the use of lower case. The menu input to HILDA is case sensitive. You will be asked at which station to calculate the cost. The station number that we have used in this example for the 3MV data is station 10.

- Type in the integer

`10`

If you type in a station number for which beam data does NOT exist you will get a message to that effect. HILDA uses the values of the parameters in the above cited module data files to obtain a cost at a station for which the beam data is defined in the data set `CalCost1.DAT`. For this example the content of these files was set with the command `SET3MV` in the instructions above.

As HILDA proceeds to calculate the minimum-cost design for the selected station it writes messages to the terminal to indicate what is happening. When it is finished with this design task it will save the results of the design on the file `CostV1.010`. The value 010 corresponds to station 10 and similarly for other station selections. It updates the file name in the data file `CostV1.DAT` so that this file name is the name of the file containing the design for the station.

In this particular example there is only one station available.

- Exit from HILDA by typing the lower case command

`stop`

- Confirm this desire to stop by typing the upper case command

`YES`

This completes the actual running of HILDA. Remember, this input is case sensitive.

If there had been more stations with their associated data sets, you could have obtained a minimum-cost design at another station. However, note that the present version of HILDA would have used the current data in the module data files. There is no way to update this data from within HILDA, for this version. The beam data would be as defined in the data file `CalCost1.DAT` and thus it would be whatever had been set for the different stations before starting the execution of HILDA. Although it is easy to edit the module data files using a text editor, future versions of HILDA will have a user interface that allows the user to modify selected data from within HILDA.

A copy of the terminal output from HILDA for this example is on the HPD Disk that is associated with this report. It is in the *HILDA/terminal/3MV* document that is in the *HILDA/Log* folder of this example. You can find the HILDA/Log folder in the *dr4MJ @ 3MV* folder. See the *Appendix* for a guide to the HPD Disk.

The Output File CostV1.NNN

The basic output file generated during a HILDA run has the name CostV1.NNN where NNN is the station to which the file corresponds. In this particular example we have produced a design at station 10; so the output file is CostV1.010. Had we requested designs at other stations, we would also have those files as HILDA output files.

These CostV1.NNN files represent the latest HILDA output HILDA. This means that we should save them if we will again calculate a design at the same station. For example, if we run HILDA twice to calculate a design at station 10, the file CostV1.010 will contain the information pertaining to the second run. This information has replaced the information for the first run; which information is lost, if it is not transferred to a differently named file.

These files are logfiles and they contain all the data used and all the results generated for the particular design that HILDA selected as having the minimum cost. Therefore, the CostV1.010 file generated by this example contains much information. The sections of this file that contain the data sets read by the HILDA modules will be better understood after those data files are described in other sections below.

In what follows we present short selections from the file CostV1.010 and describe their contents. The complete output file for this example can be found in the folder *HILDA.LOG*, which is in the *dr4MJ @ 3MV* folder. The interested reader can find these folders in the HPD disk.

The primary results of the HILDA calculation are at the beginning of the CostV1.010 file and are shown in Figure 2-2, *HILDA Output A*. The first three lines of output are for identification purposes showing the station at which the calculation was done and the date and time for the calculation. The next three lines indicate that the variable names that appear in this output have standard FORTRAN implicit naming conventions. This information is not of general interest to most users. However, this file is a logfile and is readable by the HILDA input routines. Future versions of HILDA will can read these files to analyze the output and generate reports.

Next is program output that shows the values of the HILDA parameters that give this minimum-cost design. Following this is the total cost for the station components and how much it costs to furnish one volt of energy gain.

We note that for this particular example the input data restricted the search to one point in the parameter space. However, the point that was selected for this example is the point for which HILDA found a minimum-cost design when it had a rather large grid of points over which it searched. That is why we still refer to this as the minimum-cost design, even though only one point was looked at.

Next come the input variables that define the beam at this 3MV station of the 4MJ driver. The system charge Q_{sys} gives the total Coulombs in the beam. Thus, each of the 16 beams carries 1/16th of this charge. The beam is made up of mercury ions with a charge state of 3. The undepressed tune (single beam, no space charge) is 72 degrees. The normalized emittance is .000001 π meter-radian.

This output is the primary output from the HILDA minimum-cost design calculation.

In Figure 2-3, *HILDA Output B*, we show what can be considered as secondary output from the station 10 HILDA run. The first part gives the number of Coulombs in each of the 16 beams. This is the value of the variable *tauI*, which is the beam pulse-length times the beam current. Next come dimensionless dynamic quantities for the mercury ions that make up the beam; followed by their magnetic rigidity.

The parameter grid that HILDA scanned over is then shown. For this particular example there is, as we have mentioned above, only one point in this grid. Usually there would be a parameter range that the user feels is reasonable. HILDA would then return, in this CostV1.NNN file, the information pertaining to the design that yields the minimum cost.

When there are many points in the parameter space the calculation can take a considerable amount of computing time. The time numbers shown are averages. For some points the beam cannot be transported and HILDA does not attempt to produce a design; these points take very little time. For other points a full design calculation is performed, but the design is rejected because a constraint is violated. For example, the acceleration cell might not fit into the available space. These full designs take the longest time.

Even though the numbers for the times are averages, they are useful in estimating needed computer time. We note that with only one point in the grid this example gives times for the calculation that are too long. It should be noted that HILDA reads a data file for each module that requires parameter data. However, this parameter setting is done only once for the initial point in the parameter grid.

The Figure 2-4, *HILDA Output C* contains output that can be considered to be intermediate results. Each module in HILDA performs a task that requires variable input, *IN*, and produces values for output variables, *OUT*. There are, usually, local, intermediate quantities calculated by the process. These quantities define the state of the module for the minimum-cost calculation. They are saved on this output logfile to help understand the calculation and to help insure that the design corresponds to a correct calculation.

The module *CostGrPt*, that finds the cost at each point in the parameter grid, calculated the intermediate quantities shown here. We should note that when a module generates output, that output is usually written to a file with filetype *OUT*. In this case we are looking at the content of the file *CostGrPt.OUT*. This file was copied to the file *CostV1.010* so it would be saved.

The intermediate quantities shown here come from two modules: *AlSighAbar* which solves the dynamic problem and *QIT* which calculated the shown quantities. The quantity *alpha* is a parameter relating to the beam dynamics. This is defined in the section *The HILDA Model: Beam Transport Equations* of this report. The parameter *sigmaH* is the depressed tune of the head of the beam, in each of the 16 beams. The quantity *abar* is an average beam size; again see the *Hilda Model* section of this report.

From the module *QIT* we have the perveance of the beam, the current, the pulse width (longitudinal length), the average voltage gain per meter (electric field strength), and the volt-seconds furnished by the acceleration cell.

As we have previously mentioned, HILDA modules that have significant output write that output to a file for later recovery. In Figure 2-5, *HILDA Output D* we show that part of the file *CostV1.OUT* that has been copied from the output file created by the module *ScQ30*.

In each of the HILDA module output files we record the file name, the version of the module that created the file, and the date and time of the creation of the file. We then print the I/O variables of the module.

The module *ScQ30* designs a superconducting quadrupole, so we have as its variable output two quantities. The variable *dolTArray* is the total dollar cost of the SC quadrupole array; in this case a 16 beam array. The variable *roTArray* is the outer radius of the array. The acceleration cell designed by the module *StrucCore* must know this dimension, since it must have a core inner diameter that will accommodate this quadrupole array.

The rest of the information in this *ScQ30* output pertains to the design of the superconducting quadrupole. In the section *The Module Data Files: ScQ30.DAT SC Quadrupole Design Data* we give a description of the data file, a portion of which is included here. We also have pictures of the superconducting quadrupole, see Figure 2-14, *StrucCore.DAT*. The design parameters that HILDA calculates, which are shown below, will be more meaningful when reference is made to those pictures.

The calculated parameters shown below each have a descriptive label that should help the reader to understand how they relate to the pictures in the data set. We note here that this output file is written as a readable HILDA file. Each item has as its identifier the actual *ScQ30* variable name. This file can be processed by yet to be built HILDA routines that produce input files for programs that analyze the results, or that produce design drawings of the quadrupole.

We will not, at this time, describe in detail each parameter printed in this output. We note that the output of the parameters is divided into two parts. The first part refers to one individual SC quadrupole. For this quadrupole we have radial dimensions, cross-sectional areas, volumes, weights, and costs of the components. The individual quadrupoles are then bundled together to form an array of quadrupoles. In this case the number of beams is 16 so there are 16 quadrupoles, or channels, in the array. The second part is for this quadrupole array; we again show radial dimensions, areas, volumes, weights, and costs.

Following this parameter specification part of the output comes a section that shows the actual materials used for this design. In the data file ScQ30.DAT, which is described in another section of this report, we can specify sets of material. A particular selection was made by HILDA in designing the quadrupole for this minimum cost calculation. We record here that selection, in totality, so that there will be no question later about what was used for this design. The information that appears in this output is described in the section *The Module Data Files: ScQ30.DAT SC Quadrupole Design Data* that treats the complete data file.

Figure 2-6, *HILDA Output E* is the output files created by the module StrucCore. It is similar to the previous SC quadrupole file in that it has parameters and data that define the minimum-cost design; in this case the acceleration cell design.

The primary output from this module is the variable *dolAStruct*, which is the cost of the acceleration cell. This cell does NOT include the focusing/defocusing quadrupole. This dollar amount is simply the cost of the cell as determined by its component material costs. The design information in this file should be referred to the pictures in the data file StrucCore.DAT, which is described in the section *The Module Data Files: StrucCore.DAT Acceleration Cell Design Data*.

The file contains sections of information. Again we will not describe each parameter. The first section is the dollar cost of the components of the cell. This cost is based on the cost of the material used, as obtained from the aforementioned data file. Next we show the selection parameters used in this design. These, when associated with the contents of the data file StrucCore.DAT, define the component materials. We next give component weights, followed by cell and core parameters. The cell and core parameters should be sufficient to draw a picture of the cell that HILDA has designed.

The parameters for the actual material used in this design are then recorded for future reference. Future versions of HILDA may modify this output so that the actual names of the material used are recorded in this output. Now it is necessary to use the selection flags to look up the material in the StrucCore.DAT data file.

In Figure 2-7, *HILDA Output F* we show an outline of the remainder of the CostV1.010 file. The data files used by the HILDA modules are copied and recorded in this output file. This is not necessarily redundant. HILDA is very modular and future versions of HILDA may modify the data file during execution to help HILDA find a minimum-cost design. The actual data files used are those that are recorded in this output file. A description of these data files appears in the section *The Module Data Files*.

The final entry into this logfile is a list of module versions. We have included here one of the version files for illustrative purposes. This is of no general interest to the user. However, it serves a purpose of verifying that the HILDA modules have been correctly updated. The version file must be updated when HILDA is updated and modules are changed or added. If this is not done, a message to that effect will appear in this log of the module versions. This is a redundancy check that the program modules have been carefully updated. It does not insure that this is the case, but a message that a version file does not agree with the module version should serve as a warning to the user that all may not be well in HILDA.

This also is a place where summaries of the modules, their equations, and algorithms can appear. This is presently not done, but as HILDA is updated it may prove desirable to have such a summary directly associated with the minimum-cost design. It may not always be possible to recover the exact version of the module that produced the results in this logfile.

We conclude this discussion of the HILDA output file CostV1.NNN by noting that although the output for the designed element may appear to be more than is really required this is not the case. There are many

cross-checks that can be performed using this output. To insure that data input mistakes have not been made, these checks should occasionally be done. These checks also build confidence that HILDA is indeed calculating correct results and designing reasonable elements. This file is meant to be a logfile and as such it is a record of what went into the calculation and what was produced by the calculation. Unless something is wrong, or the design criteria or logic has changed, it should be possible to recreate this output by using the printed input.

HILDA Output File CostV1.010

Figure 2-2. HILDA Output A

```

FILE: CostV1.010
DATE: 91/10/22
TIME: 16:02:51
I/O variable types:
implicit real      (a-h,o-z)
implicit integer (i-m)

The optimum value was found at:
RL      = 3.4999999E-01 : ! half-period           [m]
a       = 3.5000000E-02 : ! beam radius           [m]
delV    = 7.0000000E+03 : ! voltage gain           [V]
eta     = 4.0000001E-01 : ! packing factor         [ ]

The cost is:
costDol  = 1.6298813E+05 : ! cost                  [$]
perVoltDol= 2.3284018E+01 : ! voltage gain cost    [$ / delV]

The input variables for this solution are:
IDStation = 10 : ! station name
Qsys      = 1.3333330E-03 : ! system charge        [C]
numBeam   = 16 : ! number of beams                [ ]
Amu       = 2.0000000E+02 : ! atomic mass           [amu]
q         = 3.0000000E+00 : ! charge state          [e]
V         = 3.0000000E+00 : ! cumulative voltage    [MV]
sig0      = 7.2000000E+01 : ! undepressed tune      [deg]
epsn      = 1.0000000E-06 : ! nor. emit., no PI    [m-r]

```

Figure 2-3. HILDA Output B

```

Associated quantities are given below:

Process KenVar:
tauI      = 8.3333311E-05 : ! charge per beam      [C]
betaGamma = 9.8295826E-03 : ! beta * gamma       [ ]
gamma     = 1.0000483E+00 : ! Energy/rest Energy  [ ]
beta      = 9.8291077E-03 : ! v/c                 [ ]
Brho      = 2.0360343E+00 : ! magnetic rigidity    [T-m]

Process CostGrPt scanned over the grid:
RLmin     = 3.4999999E-01 : ! min. half period    [m]
RLmax     = 3.4999999E-01 : ! max. half period    [m]
delRL     = 3.4999999E-01 : ! grid interval        [m]
aMin      = 3.5000000E-02 : ! min. beam size       [m]
aMax      = 3.5000000E-02 : ! max beam size        [m]
dela      = 3.5000000E-02 : ! grid interval        [m]
delVmin   = 7.0000000E+03 : ! min. voltage change  [V]
delVmax   = 7.0000000E+03 : ! max. voltage change  [V]
delVdelV  = 7.0000000E+03 : ! grid interval        [V]
etaMin    = 4.0000001E-01 : ! min. packing factor [ ]
etaMax    = 4.0000001E-01 : ! max. packing factor [ ]
delEta    = 4.0000001E-01 : ! grid interval        [ ]

The time required for the grid scan:
numGrPts  = 1.0000000E+00 : ! # of grid points    [ ]
delTime   = 3.9062500E-02 : ! grid scan time      [s]
aveTime   = 3.9062500E-02 : ! ave. case time      [s]

```

Figure 2-4. HILDA Output C

```

Associated intermediate Results:

FILE: CostGrPt.OUT                      VERSION: 910910
DATE: 91/10/22 // 16:02:50
I/O variable types
implicit real    (a-h,o-z)
implicit integer (i-m)

IN
IDStation = 10      ! station for this calculation
numBeam   = 16      ! number of beams
tauI      = 8.3333311E-05 ! charge                [C]
sig0      = 7.2000000E+01 ! undepressed tune    [deg]
betaGamma = 9.8295826E-03 ! beta * gamma
beta      = 9.8291077E-03 ! v/c                 [ ]
Brho      = 2.0360343E+00 ! magnetic rigidity   [T-m]
RL        = 3.4999999E-01 ! half period         [m]
a         = 3.5000000E-02 ! beam radius         [m]
delV      = 7.0000000E+03 ! voltage gain        [V]
eta       = 4.0000001E-01 ! packing factor      [ ]

OUT
perVoltDol = 2.3284018E+01 :! voltage gain cost    [$ / delV]

AlSighAbar
alpha      = 5.8574492E-01 :!                          [ ]
sigmaH     = 5.9359894E+00 :! depressed tune        [deg]
abar       = 2.6977101E-02 :! an ave. beam size    [m]

```

Figure 2-4. HILDA Output C (continued)

```

QIT
perv      = 2.0383161E-03  !! perv
current   = 2.0049188E+00  !! beam current           [A]
taup      = 4.5720877E-05  !! pulse width           [s]
E          = 2.0000000E+04  !! ave. volt gain/HLP     [V/m]
voltSec    = 3.2004613E-01  !! volt seconds          [V-s]

ENDFILE:   CostGrPt.OUT

```

Figure 2-5. HILDA Output D

```

FILE:   ScQ30.OUT                      VERSION: 910910
DATE:   91/10/22  // 16:02:49
I/O     variable types
implicit real      (a-h,o-z)
implicit integer (i-m)

IN
a          = 3.500000E-02  !!beam edge radius           [m]
RL         = 3.500000E-01  !!half period                [m]
eta        = 4.000000E-01  !!packing factor             []
Brho       = 2.036034E+00  !!magnetic rigidity         [T-m]
sig0       = 7.200000E+01  !!undepressed tune           [deg]
numBeam    = 16           !!number of beams             []

OUT
doltArray  = 5.585644E+04  !!cost of quad array         [$]
roTArray   = 5.513123E-01  !!outer radius of quad array [m]

Data and calculated Quadrupole parameters:
sig0       = 7.200000E+01  !!undepressed tune           [deg]
rWire      = 6.965000E-02  !!inner wire radius         [m]
rWo        = 7.965000E-02  !!outer wire radius         [m]
drWire     = 1.000000E-02  !!wire thickness         [m]
BWire      = 3.972905E+00  !!field, inner wire radius [T]
BWo        = 4.543314E+00  !!field, outer wire radius [T]
Bprime     = 5.704099E+01  !!quad., field gradient   [T/m]
avJ        = 1.130178E+09  !!average current density [A/m**2]
R          = 5.375000E-02  !!aperture radius         [m]
drWrap     = 5.480373E-03  !!quad wrap, thickness   [m]
pitch      = 1.732607E-01  !!beam center to center   [m]
endMag     = 7.965000E-02  !!length of magnet ends   [m]
zlMag      = 2.993000E-01  !!length of magnet iron   [m]
wtScQuad   = 3.649901E+01  !!weight of 1 SC quadrupole [kg]
costQuad   = 3.003825E+03  !!cost of 1 SC quadrupole  [$]
wtTArray   = 8.957938E+02  !!weight of SC quad Array  [kg]
doltArray  = 5.585644E+04  !!cost of SC quad Array    [$]

SINGLE SC QUADRUPOLE ASSEMBLY
Radial dimensions
drPIC      = 2.590000E-02  !!Pipes, Insulation and Cooling [m]

Individual radii
rPipe (1)  = 5.375000E-02  !!Pipes                      [m]
rPipe (2)  = 6.765001E-02  !!Pipes                      [m]
rInsul(1)  = 5.575000E-02  !!Insulation                 [m]
rInsul(2)  = 6.005000E-02  !!Insulation                 [m]
rInsul(3)  = 6.435000E-02  !!Insulation                 [m]
rCool (1)  = 5.905000E-02  !!Cooling                    [m]
rCool (2)  = 6.335000E-02  !!Cooling                    [m]

```

Figure 2-5. HILDA Output D (continued)

Areas, xy cross-section			
xyScQ	=	3.001929E-02	!!One SC quadrupole [m**2]
xyWrap	=	1.008864E-02	!!Wrap (stress), incl. shell [m**2]
xyCable	=	4.690398E-03	!!Cable, SC + non-SC! [m**2]
xyPIC	=	6.163993E-03	!!Tot. Pipes, Insul. & Cool. [m**2]
xyVac	=	9.076258E-03	!!Vacuum, Beam pipe [m**2]
Volumes for One SC Quadrupole			
volScQ	=	8.984773E-03	!!Complete Quad. [m**3]
volWrap	=	3.019529E-03	!!Wrap (stress), incl. shell [m**3]
volCable	=	1.403836E-03	!!Cable, SC + non-SC [m**3]
volPipe	=	4.641217E-04	!!Pipes, total [m**3]
volInsul	=	1.148701E-03	!!Insulation, total [m**3]
volCool	=	2.320608E-04	!!Cooling, total) [m**3]
volPIC	=	1.844883E-03	!!Tot. Pipes, Insul. & Cool. [m**3]
volVac	=	2.716524E-03	!!Vacuum, Beam Pipe [m**3]
Weights of Components in One SC Quadrupole			
wtScCab	=	3.809955E+00	!!SC Cable material [kg]
wtCabS	=	8.032485E+00	!!non-SC Cable material [kg]
wtCable	=	1.184244E+01	!!Total Cable Weight [kg]
wtWrap	=	2.465657E+01	!!Outer Wrap, includes shell [kg]
wtPipe	=	0.000000E+00	!!Pipe layers [kg]
wtInsul	=	0.000000E+00	!!Insulation layers [kg]
wtCool	=	0.000000E+00	!!Cooling layers [kg]
wtScQuad	=	3.649901E+01	!!Total of one SC Quadrupole [kg]
Costs of Components in one SC Quadrupole			
dScCab	=	1.142987E+03	!!SC Cable material [\$]
dCabS	=	4.016242E+02	!!non-SC Cable material [\$]
dCable	=	1.544611E+03	!!Total Cable, SC + non-SC [\$]
dWrap	=	6.164143E+02	!!Wrap (stress), includes shell [\$]
The rest of the SC Quadrupole (channel)			
dPipe	=	0.000000E+00	!!Pipes [\$]
dInsul	=	0.000000E+00	!!Insulation [\$]
dCool	=	0.000000E+00	!!Cooling [\$]
dBalace	=	8.427998E+02	!!Pipes, Insulation and Cooling [\$]
costQuad	=	3.003825E+03	!!Total, one SC Quad. (channel) [\$]
COMPLETE ARRAY of numBeam SC Quadrupole channels + Wrap			
Radial dimensions			
xAScQ	=	6.930430E-01	!!Width of the SC Quads [m]
yAScQ	=	6.930430E-01	!!Height of the SC Quads [m]
dxAOtC	=	4.331519E-02	!!Array outer collar,dr [m]
dyAOtC	=	4.331519E-02	!!Array outer collar,dr [m]
Areas, xy Cross Section			
xyAScQ	=	4.803086E-01	!!SC Quadrupole bundle [m**2]
xyAVac	=	1.452201E-01	!!Vacuum, Beam Pipes [m**2]
xyAOtC	=	1.275820E-01	!!Array outer Collar [m**2]
xyAQuad	=	6.078905E-01	!!Array, Total xy Area [m**2]
Volumes			
volAScQ	=	1.437564E-01	!!SC Quadrupole bundle [m**3]
volAVac	=	4.346439E-02	!!Vacuum, Beam Pipes [m**3]
volAOtC	=	3.818528E-02	!!Array outer Collar [m**3]
volAQuad	=	1.819416E-01	!!Array Total xyz space [m**3]

Figure 2-5. HILDA Output D (continued)

```

Weights of Array Components (Total for numBeam SC Quads.)
wtAScCab = 6.095929E+01  :!SC Cable material [kg]
wtACabS = 1.285198E+02  :!non-SC Cable material [kg]
wtACable = 1.894790E+02  :!Total Cable Weight [kg]
wtAWrap = 3.945052E+02  :!Outer Quad. Wraps (stress) [kg]
wtAPipe = 0.000000E+00  :!Pipes [kg]
wtAInsul = 0.000000E+00  :!Insulation layers [kg]
wtACool = 0.000000E+00  :!Cooling layers [kg]
wtAScQ = 5.839842E+02  :!SC Quadrupoles [kg]
wtAOtC = 3.118096E+02  :!Outer Array collar [kg]
wtTArray = 8.957938E+02  :!Total of the Complete Array [kg]

Costs of Array Components (Total for numBeam SC Quads.)
dAScCab = 1.828779E+04  :!SC Cable material [$]
dACabS = 6.425988E+03  :!non-SC Cable material [$]
dACable = 2.471377E+04  :!Total for cable, SC + non-SC [$]
dAWrap = 9.862629E+03  :!Quad Wraps (stress) [$]
dAPipe = 0.000000E+00  :!Pipes [$]
dAInsul = 0.000000E+00  :!Insulation layers [$]
dACool = 0.000000E+00  :!Cooling layers [$]
dABalance = 1.348480E+04  :!Total for Pipes, Insul., Cool. [$]
dAQuad = 4.806120E+04  :!Total for the SC Quads [$]
dAOtC = 7.795239E+03  :!Outer Array collar [$]
dolTArray = 5.585644E+04  :!Total for Array Assembly [$]

DATA LOADED FROM FILE ScQ30.DAT

CHOOSE MATERIALS for the SC Quadrupole
iScCab = 1  :! SC wires []
iCabS = 1  :! non-SC wire space []
iWrap = 1  :! Wrap (stress) []
iPipe = 1  :! vac. & outer Quad pipes []
nPipe = 2  :! # of pipes []
drPipe = 2.000000E-03  :! thickness of pipes [m]
iInsul = 1  :! insulation layers []
nInsul = 3  :! # of layers []
drInsul = 3.300000E-03  :! thickness of layers [m]
iCool = 1  :! cooling layers []
nCool = 2  :! # of layers []
drCool = 1.000000E-03  :! thickness of layers [m]

CHOOSE MATERIAL for the Array of numBeam Quadrupoles
iAWrap = 1  :! outer wrap (collar) []

SET LIMITS and PARAMETER VALUES

Quadrupoles
drWmin = 1.000000E-02  :! minimum wire thickness [m]
drWmax = 1.000000E-01  :! maximum wire thickness [m]
dShell = 1.500000E-03  :! outer shell thickness [m]

Non-Quadrupole free space
zQend = 0.000000E+00  :! end packaging for the quads. [m]
fZSpace = 1.000000E-01  :! non-magnet space limit, frac.of L []

Coefficients for the SC Quadrupole Calculation:
cR1 = 1.250000E+00  :! aperture radius = R = cR1*a+cR2 []
cR2 = 1.000000E-02  :! [m]

Quadrupole Wrap (stress) scaling parameters
sWrap = 1.000000E-02  :! wrap thickness for BWrap, rWrap [m]
BWrap = 5.000000E+00  :! field for scaling the Quad. Wrap [T]
rWrap = 1.200000E-01  :! radius used with BWrap [m]

Quadrupole SC wire inner radius limit
rWtRL = 5.000000E-01  :! rWire .lt. eta*RL [m]

```

Figure 2-5. HILDA Output D (continued)

```

Array of numBeam Quadrupoles
fCollar = 2.500000E-01  !! wrap (collar) width, frac. of pitch {}

Quadrupole (channel) Assembly-Complexity Cost factor
BFactor = 3.900000E-01  !! times the Quad (Wrap + Cable) cost {}

Array Assembly-Complexity Cost factor
cAFact = 1.000000E+00  !! times the Array cost {}

MATERIAL USED in the SC Quadrupole Array
Superconducting Cable
iScCab = 1  !! ID # of data set
IDScMat = 'NbTi Niobium Titanium'  !!
denScMat = 7.600000E+03  !! material density [kg/m**3]
uScMat = 3.000000E+02  !! unit cost [$ /kg]
qFScMat = 1.000000E+00  !! quantity factor {}
cFScMat = 1.000000E+00  !! complexity factor {}
Current density parameters and Field limits
cJCoeff = 2.900000E+09  !! slope of cJ curve [A/m**2]
cJBn = 1.000000E+01  !! numerator B parameter [T]
cJBd = 5.000000E+00  !! denominator B parameter [T]
rLamda = 3.571000E-01  !! S C wire pack. fraction [T]
BWomax = 1.000000E+01  !! B at outer wire, maximum [T]

Non SC Cable
iCabs = 1  !! ID # of data set
IDCMat = 'Cu Copper'  !!
denCMat = 8.900000E+03  !! material density [kg/m**3]
uCCMat = 5.000000E+01  !! unit cost [$ /kg]
qFCMat = 1.000000E+00  !! quantity factor {}
cFCMat = 1.000000E+00  !! complexity factor {}

Quadrupole Wrap (Stress)
iWrap = 1  !! ID # of data set
IDWMat = 'Steel'  !!
denWMat = 8.165700E+03  !! material density [kg/m**3]
uCWMat = 2.500000E+01  !! unit cost [$ /kg]
qFWMat = 1.000000E+00  !! quantity factor {}
cFWMat = 1.000000E+00  !! complexity factor {}

Array Outer Wrap (Collar)
iAWrap = 1  !! ID # of data set
IDAotMat = 'Steel'  !!
denAotMat = 8.165700E+03  !! material density [kg/m**3]
uCAotMat = 2.500000E+01  !! unit cost [$ /kg]
qFAotMat = 1.000000E+00  !! quantity factor {}
cFAotMat = 1.000000E+00  !! complexity factor {}

Pipes , Vacuum and around Quad outer insulation layer
iPipe = 1  !! ID # of data set
IDPMat = 'No Data on this Material'  !!
denPMat = 0.000000E+00  !! material density [kg/m**3]
uCPMat = 1.000000E+00  !! unit cost [$ /kg]
qFPMat = 1.000000E+00  !! quantity factor {}
cFPMat = 1.000000E+00  !! complexity factor {}

Insulation (thermal)
iInsul = 1  !! ID # of data set
IDIMat = 'No Data on this Material'  !!
denIMat = 0.000000E+00  !! material density [kg/m**3]
uCIMat = 1.000000E+00  !! unit cost [$ /kg]
qFIMat = 1.000000E+00  !! quantity factor {}
cFIMat = 1.000000E+00  !! complexity factor {}

```

Figure 2-5. HILDA Output D (continued)

```

Cooling Layers, data sets
iCool      = 1           :! ID # of data set
IDClay     = 'No Data on this Material'      :!
denCLay    = 0.000000E+00 :! material density      [kg/m**3]
uCClay     = 1.000000E+00 :! unit cost              [$ /kg]
qFCLay     = 1.000000E+00 :! quantity factor       []
cFCLay     = 1.000000E+00 :! complexity factor      []
ENDFILE:   ScQ30.OUT

```

Figure 2-6. HILDA Output E

```

FILE:  StrucCore.OUT                      VERSION: 910910
DATE:  91/10/22  //  16:02:50

I/O  variable types
implicit real    (a-h,o-z)
implicit integer (i-m)

IN
  voltSec      = 3.2004613E-01      :!volt-sec per half period  [V-s]
  numBeams     = 16                 :!number of beams         [ ]
  roTArray     = 5.5131233E-01      :!outer radius, quad array [m]
  voltGain     = 6.9999998E-03      :!peak acc. per half period [MV]
  halfPeriod   = 3.4999999E-01      :!lattice half-period length [m]

OUT
  dolAstruct   = 1.071317D+05      :!HLP cost   of acc. structure [$]
The intermediate values calculated and returned are:
COST OF COMPONENTS
  numdolHlp    = 6                 :!number of costs returned
  dolAm        = 3.180715D+04      :!core amorphous material    [$]
  dolCorH      = 1.443943D+04      :!core housing                [$]
  dolCelH      = 2.573705D+04      :!cell housing                [$]
  dolCCI       = 3.896130D+03      :!core/cell housing insulation [$]
  dolGI        = 3.107991D+04      :!gap insulator               [$]
  dolDiC       = 1.720037D+02      :!dielectric coolant          [$]
MATERIALS USED: flags select material from StrucCore.dat
  numidMat     = 8                 :!# of used material flags returned
  idAm         = 1                 :!core winding amorphous tape
  idWTape      = 4                 :!core tape width
  idCSM        = 2                 :!core submodule housing
  idCH         = 2                 :!cell housing
  idIN         = 2                 :!core/cell insulation
  idDiC        = 4                 :!dielectric coolant
  idGP         = 6                 :!gap vacuum pressure range
  idGI         = 1                 :!gap insulator

COMPONENT WEIGHTS:
  numwtHlp     = 20                :!# of weights that are returned

Half Lattice Period weights for the acceleration structure
  wtAstruct    = 1.078513D+04      :!HLP weight of acc. structure [kg]
  wtAm         = 6.361431D+03      :!core amorphous material     [kg]
  wtCorH       = 1.458720D+03      :!core housing                 [kg]
  wtCelH       = 2.600044D+03      :!cell housing                 [kg]
  wtCCI        = 1.887504D+02      :!core/cell housing insulation [kg]
  wtGI         = 7.256784D+01      :!gap insulator                [kg]
  wtDiC        = 1.036167D+02      :!dielectric coolant           [kg]

```

Figure 2-6. HILDA Output E (continued)

Sub	component	weights		
wtCore	=	6.361431D+03	!!amor. material, per core	[kg]
wtCEP	=	4.528662D+02	!!housing end plate, per core	[kg]
wtCOH	=	3.786480D+02	!!outer housing, per core	[kg]
wtCOB	=	1.743392D+02	!!inner bobbin, per core	[kg]
wtCHPC	=	1.458720D+03	!!housing total, per core	[kg]
wtCEI	=	5.450253D+01	!!end plate ins. (2 per core)	[kg]
wtCOI	=	2.106926D+01	!!outer housing (ins./core)	[kg]
wtCI	=	1.300743D+02	!!insulation total, per core	[kg]
wtCHIE	=	5.867611D+01	!!cell hous. (end ins. 1/core)	[kg]
wtCHI	=	5.867611D+01	!!cell hous. insulation, total	[kg]
wthSGEP	=	2.083088D+03	!!cell hous. end plate, pair	[kg]
wtOHR	=	4.619303D+02	!!cell hous. supp. ring	[kg]
wthSGGI	=	5.502587D+01	!!gap insulator support ring	[kg]

CELL/CORE PARAMETERS:

numCP	=	29	!!# of parameters returned	
numCore	=	1.000000D+00	!!# of cores & PFNs per cell	[]
cellL	=	2.936568D-01	!!z axis, cell length	[m]
gIL	=	1.666667D-01	!!gap insulator length	[m]
pV	=	2.000000D-01	!!required PFN peak per core	[MV]
tI	=	3.000000D-02	!!gap insulator width	[m]
QAl	=	5.634315D-02	!!support lgth., quad., etc.	[m]
zIH	=	2.000000D-02	!!cell housing widths, z axis	[m]
zPH	=	1.000000D-02	!!core housing widths, z axis	[m]
zICi	=	1.015228D-02	!!z length, core insulator	[m]
delTC	=	2.000000D-02	!!core housing top	[m]
delBC	=	2.000000D-02	!!core housing bottom	[m]
delCH	=	2.309401D-02	!!cell housing top	[m]
riH	=	6.064436D-01	!!inside rad. to acc. gap insul.	[m]
delRg	=	5.128205D-02	!!radial cell-acc. gap	[m]
riM	=	6.877257D-01	!!core housing inner radius	[m]
riC	=	7.077256D-01	!!core inside radius	[m]
roC	=	1.495241D+00	!!outside radius of core	[m]
roCore	=	1.525393D+00	!!core housing outer radius	[m]
roHSG	=	1.548487D+00	!!cell housing outer radius	[m]
acAM	=	1.280185D-01	!!acc. cell amor. mat. area	[m**2]
pFam	=	8.000000D-01	!!packing fraction	[]
aC	=	1.600231D-01	!!core cross sectional area	[m**2]
wTape	=	2.032000D-01	!!core amor. mat. tape width	[m]
hC	=	7.875151D-01	!!height of amorphous material	[m]
aRC	=	3.875566D+00	!!core amor. mat. h/w ratio	[]
fQ	=	1.100000D+00	!!quad array per. supp. factor	[]
gILmax	=	2.349255D-01	!!max. gap ins. length	[m]
aRCmax	=	4.000000D+00	!!max. hth/width ratio, core	[]
QAlmin	=	3.500000D-02	!!min. quad. support length	[m]

MATERIAL DATA Values actually used:

Core amorphous material

denAm	=	7.180000E+00	!!density	[gm/cm**3]
cFacAm	=	1.000000E+00	!!complexity factor	[]
qFacAm	=	1.000000E+00	!!quantity factor	[]
uCostAm	=	5.000000E+00	!!unit cost	[\$/kg]
pFam	=	8.000000E-01	!!radial packing fact.....	[]
delBAm	=	2.500000E+00	!!flux swing	[T]

Core tape widths

wTape	=	2.032000E+01	!!core tape width	[cm]
aRCmax	=	4.000000E+00	!!max. height/width	[]

Figure 2-6. HILDA Output E (continued)

```

Core Sub Module housing material
denCSM = 8.165700E+00      :!density          [gm/cm**3]
cFacCSM = 1.000000E+00      :!complexity factor      []
qFacCSM = 1.000000E+00      :!quantity factor        []
uCostCSM = 9.898700E+00      :!unit cost              [$ /kg]

Cell Housing material
denCH = 8.165700E+00      :!density          [gm/cm**3]
cFacCH = 1.000000E+00      :!complexity factor      []
qFacCH = 1.000000E+00      :!quantity factor        []
uCostCH = 9.898700E+00      :!unit cost              [$ /kg]
emCH = 3.000000E+01      :!elasticity mod. [10**6 lb/in**2]

Core & cell Housing Insulation
denIN = 9.850000E-01      :!density          [gm/cm**3]
cFacIN = 1.000000E+00      :!complexity factor      []
qFacIN = 1.000000E+00      :!quantity factor        []
uCostIN = 2.064170E+01      :!unit cost              [$ /kg]
bVoltIN = 1.970000E+02      :!break down voltage    [kV/cm]

Gap Insulator
denGI = 3.717000E+00      :!density          [gm/cm**3]
cFacGI = 1.000000E+00      :!complexity factor      []
qFacGI = 1.000000E+00      :!quantity factor        []
uCostGI = 4.282876E+02      :!unit cost              [$ /kg]
bVoltGI = 1.200000E+01      :!break down voltage    [kV/cm]

Dielectric coolant
denDiC = 1.800000E+00      :!density          [gm/cm**3]
cFacDiC = 1.000000E+00      :!complexity factor      []
qFacDiC = 1.000000E+00      :!quantity factor        []
uCostDiC = 1.660000E+00      :!unit cost              [$ /kg]
bVoltDiC = 3.900000E+01      :!break down voltage    [kV/cm]

Acc. Gap Voltage break down strength
bVRGap = 0.500000E+02      :!                    [kV/cm]

ENDFILE:  StrucCore.OUT

```

Figure 2-7. HILDA Output F

```
DATA FILES USED :  
  
FILE:  AlSighAbar.DAT  
Image of the file used for the minimum-cost design.  
ENDFILE: AlSighAbar.DAT  
  
FILE:  ScQ30.DAT  
Image of the file used for the minimum-cost design.  
ENDFILE  ScQ30.DAT  
  
FILE:  TranMod.DAT  
Image of the file used for the minimum-cost design.  
ENDFILE: TranMod.DAT  
  
FILE:  StrucCore.DAT  
Image of the file used for the minimum-cost design.  
END:    StrucCore.DAT  
  
PROCESS VERSIONS:  
c  FILE: KenVar.EQU                      VERSION: 910910  
c      SUBROUTINE KenVar  
c      version = '910910'  
c  ENDFILE: KenVar.EQU  
  
ENDFILE:  CostV1.010
```

The Module Data Files

Each HILDA data file discussed below is associated with a HILDA module of the same name. The module will read the file *module name.DAT* to get the data it requires. For example, the module *CostV1* reads the file *CostV1.DAT*. As mentioned previously, the data sets should have enough comments to explain themselves. We limit ourselves to comments on their contents.

CalCost1.DAT Beam Parameter File

At each station that the user intends to use HILDA to find a minimum-cost design it is necessary to define the beam parameters. Those parameter values are furnished in this data file, which is subsequently read by the HILDA module *CalCost*. In Figure 2-8, *CalCost1.DAT* below we show the contents of this file. As we have previously noted, the HILDA I/O routines can read a fully formatted ASCII data file. The file shown in Figure 2-8 is a valid HILDA data file, when written as a flat ASCII text file. This means that the file does not contain any special word processor information, such as special characters that define boldfaced or outlined text. The flat ASCII text files of this example are in the folder *dr4MJ @ 3MV* that is in the folder *TXT/Hilda/DAT* of the HPD disks; see the *Appendix* for a guide to these disks.

This naming convention is adhered to throughout the folders on the HPD disks. Those folders that have MSW in their names contain Microsoft Word documents. Those that have TXT in their names contain the same files, but these are ASCII text files with no special formatting or graphic pictures. These ASCII text files can be read by HILDA.

The user should now read the file in Figure 2-8. We emphasize that these example data files are meant to be self-explanatory, so we add some comments rather than repeating the explanations that are in the data file.

In the table of parameter values shown in Figure 2-8 the quantity *varName* refers to the name of the variable that receives the value in the column *varValue*. The convention in the HILDA data files is that the actual parameter name that is used in the module appears in the data to identify the parameter that is being set. In the particular data set that we are describing the name is used. In some HILDA data sets this name is simply

a descriptive identification of the parameter. As a rule, or convention, the names used in these example data sets should NOT be changed. The numerical value assigned to the name may, of course, be changed to reflect the user's data.

The same comment applies to the *data type*; it should NOT be changed. The information in the *comment*, which comes after the ! character, is there to describe the variable.

In the particular case at hand we have identified the station to be number 10. There is no significance to this number, other than to correlate the HILDA I/O files with the station at which the user requests a minimum-cost design. However, future versions of HILDA should use this information to load the correct data into the module data files.

The total amount of charge in the beam at this station is 0.00133333 Coulombs. This is transported in 16 beams. The particles that make up the beam have 200 atomic mass units and are ions with a positive charge state of 3. At this station the cumulative acceleration voltage that the machine has supplied is 3 million volts. This does NOT mean that the energy of the beam particles is 3MeV. In this particular example it would be 9MeV, because of the +3 charge state. The undepressed tune (single particle, no space charge) of the beam is 72 degrees. The normalized emittance of each beam in the 16 beams is 0.000001 meter-radian. This is the emittance without the factor of π that is sometimes included when specifying beam emittance.

All the quantities have the units that appear in the data files. These units appear as comments, however they should always be included in the data files. Without the units it is not always possible to resolve exactly what the quantity represents.

In the file CalCost1.DAT we also have a file name, which in this example is shown as *CostV1.010*. The name furnished before the run is immaterial. In some data files you may see the file name set to null. HILDA enters the name of the file that contains the output for the minimum-cost design at the station. In this case the name will become CostV1.010. This information may not seem useful to the user, at this time. However, future versions of HILDA can make use of this information when processing run output.

CostV1.DAT Parameter Search Space

This data file, which is shown in Figure 2-9, *CostV1.DAT*, is associated with the module CostV1 that does the parameter space scan. HILDA has the free parameters: *RL* the structure half-period, *a* the maximum beam radius, *deIV* the voltage gain that the particular station supplies, and *eta* the quadrupole packing fraction. At each point (*RL,a,deIV,eta*) in this space HILDA determines if the beam can be transported. If it cannot be transported, then HILDA will NOT try to find a minimum-cost design; it will skip to the next point in the parameter space. The data file CostV1.DAT defines the parameter space by setting the limits and the number of points for each parameter.

The parameter space is scanned from minimum to maximum for each parameter. One or more points must be specified for each parameter, no parameter may be omitted. If there is one point, then HILDA will use the minimum value. If there are two points, HILDA uses the minimum and the maximum. Three or more points divide the parameter interval [min,max] into equal subintervals. For example, specifying three points will select points at the minimum, the center, and the maximum of the parameter interval.

It should be noted that the module CostV1 has four loops that are the same as shown in the data set. The inner most loop is *eta*, the next outer loop is *deIV*, then comes *a* followed by *RL*. This order is usually of no concern to the user. However, it should be kept in mind that for the current station HILDA will set the structure half-period, the beam size, and the voltage gain, and then search through the possible quadrupole packing fractions. Upon completion of this scan it will go to the next voltage gain and repeat the process. It may be that for some situations this is not the best search order.

As we mentioned before, the contents of the logfile CostV1.LOG can vary. This is controlled by setting *iLog* in this data file.

We note here that the inclusion of the comment defining the variable *IDStation* identifies this data as belonging to station 10. Future versions of HILDA will probably have this information included as data in this data file, thus eliminating the need to set up the data file for each station. Experience has shown that the useful scan space is not necessarily the same for all stations. Experience has also shown that it is often the case that there are really not many points for which the beam can be transported, when the parameter grid is allowed to span a large range of values. The logfile can be examined to determine those points at which it may make sense to do a parameter scan using a finer grid.

AlSighBar.DAT Beam Dynamics Parameters

HILDA solves equations that pertain to the dynamics of the beam transport problem. An interested user can find information pertaining to the model in the section *The HILDA Model: Beam Transport Equations* of this report. We note here that the parameters in this data set, shown in Figure 2-10, *AlSighAbar.DAT* relate to the solution of the transcendental equations that HILDA must solve; also that the limits on the depressed tune are set here. These parameters are not usually changed. This data set is meant to be self-explanatory when reference is made to the module AlSighAbar and the beam transport equations. As noted in the data set, there is usually no reason for the user to modify this data.

TranMod.DAT Transport Module Selection Data

HILDA first asks whether the beam can be transported. For those points in the parameter space for which the beam CAN be transported, HILDA designs appropriate elements. The module TranMod reads data that selects which of the available design modules to use. Presently the selection is somewhat limited: HILDA can design an iron quadrupole and it can design a superconductivity quadrupole. However, future versions will have a wider selection of design modules. The contents of the TranMod.DAT data file are shown below in Figure 2-11, *TranMod.DAT*. The data set is self-explanatory.

FeQ20.DAT Fe Quadrupole Design Data

The HILDA module FeQ20 designs an iron quadrupole. This data set, shown in Figure 2-12, *FeQ20.DAT*, furnishes all the necessary design information. The first coefficients are parameters that determine the size of the vacuum pipe enclosing the beam. We note that the beam radius a that appears in this formula is the same beam radius parameter that HILDA uses as a free parameter. If it is desired for HILDA to cost a machine for which the vacuum pipe radius is a fixed size, these coefficients must be adjusted appropriately. This adjustment was made when HILDA was run on the ILSE example that is included on the HPD disks.

The remaining coefficients are self-explanatory; the user should refer to the picture in the data set to understand their significance. We note that when this data set is written as a flat ASCII file, the graphic that constitutes the picture showing the FeQ20 quadrupole will not be written. The data set, without this graphic, is readable by HILDA.

This particular design module reads only one material set; the Fe material used for both the quadrupole poles and the return yoke. We furnish the density and the unit cost of this material. The two other factors, *cCost* and *qCost*, can be used to adjust the final cost of the quadrupole. The quantity factor *qCost* is meant to reflect the fact that producing these quadrupoles in large quantities can cause the cost to change. Likewise the complexity factor *cCost* can be used to adjust the unit cost for the cost of fabricating a complex item. Both of these factors multiply the cost arrived at when only the unit cost of the material is used. In the present example they have been set to 1 and thus have no effect on the final cost.

The picture in Figure 2-12 is a schematic transverse view of the quadrupole. The quadrupole is cylindrically symmetric about the longitudinal z axis. The labels used are the actual variable names used in the module FeQ20. The output from FeQ20 appears in the final design that is written to the logfile CostV1.010 and the parameters can be keyed to this picture.

ScQ30.DAT SC Quadrupole Design Data

The module ScQ30 designs a superconducting quadrupole. This module is similar to the module FeQ20. The ScQ30.DAT data set furnishes information needed for the superconducting quadrupole. This data set is shown in Figure 2-13, *ScQ30.DAT*.

The parameters in the section *CHOOSE MATERIALS for the SC Quadrupole* are used to select the materials for the named components of the SC quadrupole. The three pictures in this data set can be used to identify the various components. In this example the parameters have been set to select the superconducting wire as NbTi, the material filling the remainder of the winding space as Cu, and the material around the coils as steel. We then have the material for the vacuum pipe and the outer containment pipe. We also can specify data for the thermal insulation layers and for the cooling layers.

The module ScQ30 can read up to 10 data sets for each of these six items. In this particular example we only have 1, out of the possible 10 that can be read. The material flags are used to load the appropriate material data set. For example, specifying 1 for the SC wire material causes the module to load the first material data set. The actual data sets are at the end of this data file.

Note that when we specify the material for the vacuum and outer quadrupole containment pipes we also specify the number of pipes and the thickness of the pipes; all pipes have the same thickness. This is also true of the insulation and cooling layers. We specify not only the material but also the number of layers and the layer thickness. In this example no material costing data has been furnished for the pipes, insulation, and cooling layers. However, ScQ30 must have a data packet to read. Note that for these items the material density has been set to 0.0 [kg/\$] and the cost factors to 1.0. This effectively eliminates the material from the cost calculation; it adds no weight and contributes nothing to the cost.

The SC quadrupoles are bundled together to form an array of quadrupoles. In the particular example at hand we specified that there would be 16 beams, so there will be 16 quadrupoles in the array. This array has an outer wrap. The material for this outer wrap is selected next.

We then specify limits on the minimum and maximum thickness of the SC wire. The actual thickness of this wire will be determined by ScQ30; within these limits. If the module finds that a thickness less than the minimum will suffice, the wire will be set to the minimum thickness. If the module determines that the thickness needs to be greater than the maximum, the design will be flagged as unacceptable for this reason.

The complete array has an outer shell and that shell has a thickness set by the parameter *dShell*. The physical length of the magnet can be adjusted using the parameter *zQend*. It is necessary to have a certain amount of free space within the half-period. This minimum space is determined by *fZSpace*, which in this example is set to 10% of the half-period length.

The next coefficients determine the radius of the vacuum pipe; they are used in the same way as in the iron quadrupole design. Again, if it is desired to have a constant vacuum pipe size it may be necessary to adjust these parameters accordingly.

The scaling parameters should NOT be changed, unless the user understands how they are used in the module ScQ30. The radius limit parameter keeps the quadrupole from being either too short, or too large in bore. The currently set value keeps the quadrupole from having an inner SC wire radius that is larger than 1/2 the magnetic length of the quadrupole.

The final parameter determines the thickness of the wrap around the whole array. This is presently set to 1/4th the pitch of the beam. The beam pitch is the beam-center to beam-center spacing of the beams. We treat only square arrays and this is the x, or y, spacing; not the diagonal beam spacing.

Next data is furnished for some cost parameters. The parameter *BFactor* is used to reflect the material cost and is added to the cost of the quadrupole. This factor reflects the fact that it costs something to assemble the individual quadrupoles. The parameter *cAFact* is used to make it possible to isolate the assembly cost of the array from the material cost in the cost calculation. For this example it is set to 1 and has no effect.

The three figures in this data set present a transverse view, an end view of a single SC quadrupole as designed by ScQ30, and an end view of the quadrupole array. The particular example at hand has 16 beams, a square array of 4 x 4. In the pictures we have shown a square array of 4 beams. The only difference is that the 16 beam case has more quadrupoles bundled into the array. Again, the labeling uses the variable names used in the module ScQ30. This should allow the user to understand both this data set and the minimum design output written to the HILDA output files.

After the figures are the data sets for the materials that comprise the components of the SC quadrupole. The first of these is for the superconducting cable. The first parameter indicates how many of these data sets there will be; up to 10 are allowed. The cable material is then identified using a FORTRAN character constant. Note that this is a string of text enclosed in the single quotation marks. Next come the material density, the unit cost of the material, and then the quantity and complexity factors. These latter two factors play the same role here as in the FeQ20 data. They multiply the material cost of the superconducting cable. Next is a list of four values for parameters pertaining to the superconducting material. These values should not be changed, unless there use in the module is completely understood. The final parameter for this data packet is the maximum allowable field allowed at the outer SC wire radius. If the field needed is larger than this value, the design will not be accepted.

The information furnished for the non- SC Cable, the quadrupole wrap material, array outer wrap material, pipe material, insulation material, and cooling layers is identical in nature to the similarly named parameters in the SC cable data packets. We specify the number of data packets, less than or equal to 10; the identification of the material, the material density and unit cost, and then the quantity and complexity factors. In each case these later two factors are applied to the named quantity; e.g., the cooling layer costs if they appear in that data packet.

The parameter names in the data are the same as the variable names in the module that receive these values. The order and number of parameters in this data set should not be changed. However, more data packets can be added to the material data sets, provided that the number of data sets for a particular material is correctly specified. When more data packets are added the comments at the beginning of this data file should be updated to reflect the extended choice of materials.

StrucCore.DAT Acceleration Cell Design Data

The module StrucCore designs an acceleration-transport module. The basic data for this module is in the file StrucCore.DAT. This data set, an example of which is given below in Figure 2-14, *StrucCore.DAT*, is meant to be self-explanatory. The explanation that follows assumes that the reader will read the data set and will refer to the pictures in that data set. We will key the this explanation to the headings in that data set.

SET ERROR LOGGING TO TERMINAL

HILDA tries to design an acceleration module within certain constraints. For example, the length of the module must fit into the available space and the cores must not be too large in radii. When it fails to find a design it will skip to the next parameter point for which a design has been requested. The module StrucCore can inform the user that the design of the acceleration module has failed, or it can remain silent. This flag is used to make that choice.

SELECT COMPONENT MATERIALS

The acceleration structure consists of a number of basic components. These are shown in the picture *Acceleration Structure*. For each of these basic components a choice is made for the construction material.

In the data that follows we make the material choice. The data packets that define the material characteristics follow later in the data set. It is, of course, not correct to choose a material for which data does not exist. As we have previously mentioned, later versions of HILDA will have a user interface that simplifies the setting of these selection parameters.

The induction core is wound using amorphous tape. There are available in this data set two specific tape materials and these tapes come in different widths. The selection is made in the sections labeled:

CORE amorphous tape materials available

CORE amorphous tape widths available

We see from the picture that the cores are stacked longitudinally to form the acceleration module, which we also refer to as a cell. The materials that are used for making the core housings and the acceleration cell must be specified. In this particular example there are ten materials available. For the cores we have selected low carbon steel (welded & machined - 1020) as the material to use for the power lead, outer housing, and inner bobbin. The cell housing uses the same material; this includes the end plates and inner bobbin of the acceleration cell.

INSULATION/DIELECTRIC MATERIAL

There are six insulation/dielectric materials available. This material is used as shown in the pictures. The gap insulator has on one side a vacuum and on the other a dielectric coolant material. We have chosen the core and cell housing insulation to be polyethylene, the dielectric coolant to be Freon, and the area inside the gap insulator to be a vacuum. The principal physical parameter of interest for these materials is the voltage breakdown strength.

GAP INSULATOR MATERIAL.

The insulator for the module is a large concentric ring. In the present data set there is only one material for this insulator. However, in principal more material data sets could be furnished.

PULSE FORMING NETWORK

This information is used to determine the number of acceleration cores that are needed at the current station. HILDA knows the required voltage gain, ΔV . It will use as many cores as it needs to obtain this voltage gain. The maximum volts available from each core is supplied by this parameter. The present version of HILDA does not determine the cost of the pulse forming network. Future versions will include this cost and will make more extensive use of this parameter.

MINIMUM LENGTH AVAILABLE FOR QUADRUPOLE SUPPORT, ETC.

The acceleration cell must fit into the available space. There must be free space left for other items; in this example we have specified that 10% will be left open. This means that the design will not be accepted, if the cell length exceeds 90% of the half-lattice period length.

MAXIMUM AVAILABLE GAP INSULATOR LENGTH

The longitudinal length of the gap insulator is determined by the maximum voltage gain and the voltage breakdown strength of the inner space, in this case vacuum. If the required length is too long, in this case greater than 80% of the total length of the acceleration cell, the design will be rejected.

HOUSING THICKNESSES

The parameters that are set here depend, to a large degree, on the size and weight of the structures. Future versions of HILDA could have the capability of setting them appropriately, after the weights are known. Presently they are preset by the knowledgeable user. The items to which they refer are shown in the picture *Acceleration Structure*. Note that the cell housing thickness parameter zIH refers to the end plates and inner bobbin of the cell housing. The core housing parameter $zIPH$ refers to the thickness of the core end plates, one of which can be considered to be part of the power lead. The next three parameters determine the thickness of the concentric rings that make up the top of the core, the bottom or inner bobbin of the core, and the top of the cell housing. These are scaled from the values of the parameters zIH and $zIPH$ as indicated in the data set.

At this point we have selected all the component parameters for the design of the acceleration cell. The data that follows is, to a large extent, fixed in nature.

ELASTICITY SCALING FACTOR

This is NOT to be changed by the user. It pertains to a scaling formula used in StrucCore that allows for the use of materials with different elasticity.

NUMBER OF BEAMS DATA

The radial size of the acceleration cell depends on the number of beams that it must encompass. This size in turn influences the thickness of the support structures. The data furnished here should only be changed by a user who understands the use of these parameters inside the module StrucCore.

MATERIAL PROPERTIES

The data that follows pertains mostly to the materials used in the acceleration cell. The information supplied refers to the cost of the materials and to physical properties that StrucCore needs in order to use the materials in the cell and module design.

AMORPHOUS CORE TAPES and AMORPHOUS TAPE WIDTHS

The individual packets of data pertain to the properties of the available materials. In these data sets the *unit cost* refers to the material cost. The *complexity* and the *quantity* factors multiply the unit cost to take into account other costs associated with the use of the material. Note that these are associated with the material, not with the component that uses the material. In the present version of HILDA the cost of the cell is the cost of the components that make up the cell. The cost of the components is determined by the cost of the constituent material. The cost of the material is the unit cost multiplied by the aforementioned factors.

We also see that the materials can have associated properties. In particular, the amorphous tape has a packing factor and a maximum flux swing. The packing factor relates to the winding of the cores; i.e., what fraction of the core the amorphous tape occupies.

The maximum flux swing is used in StrucCore to determine the cross-sectional core area. In this sense this parameter is not really a limit. It is assumed in StrucCore that the cores will be run with this flux swing and the total core cross-sectional area is set to give the needed acceleration volts. If a certain cross-sectional area, core size, is desired then this flux swing must be appropriately set. This was done when HILDA was run on the ILSE parameters; in that example, which is on the HPD disks, the core sizes were already determined. For the present version of HILDA, finding the maximum flux swing that gives a specified core size is non-trivial. This use of the flux swing parameter may be changed in future versions of HILDA.

The tape widths available have an associated height-to-width ratio. This maximum is used to keep the cores from being too large in diameter. HILDA stacks acceleration cores side-by-side, but not vertically. The needed cross-sectional area is obtained by stacking cores longitudinally and by making their outer radius as large as necessary. If this height-to-width ratio is exceeded the design is rejected. For example, the 5 cm tape cores should not be more than 50 cm thick. These numbers are material dependent and would not usually be changed by a user, once they are set for the specified material.

CORE MODULES and CELL HOUSING MATERIALS

The material data packets that follow all have the same type of information. The parameter *emMat* is the elasticity of the material in the units that are shown.

STRUCTURE INSULATION/DIELECTRIC MATERIALS

The physical parameter of interest here is the material's voltage breakdown strength. StrucCore determines the thickness, or length, of the insulating material by assuming that the material will be run at the voltage breakdown number. For example, the thickness in the cores and cell of the polyethylene insulation depends on the 197 kV/cm voltage breakdown value furnished in this data set. This direct use of the voltage breakdown parameter means that any safety factors that a user feels are necessary should be included in this number.

The units for all the parameter values are as indicated in the data set. Any unit conversions needed to agree with the HILDA convention of using MKS units for most of the calculations are done in the module StrucCore after it has read the parameter data.

Figure 2-8. CalCost1.DAT

```

c  FILE:  CalCost1.DAT
c  DATE:  910922
c  EXAMPLE:  dr4MJ@3MV
      At each station this data defines the basic parameters of the beam that
      is being transported. Upon completion of the cost minimization calculation
      at the station, the file CostV1.nnn, where nnn is the value of IDStation,
      will contain all the design information.
Basic Beam Data for 4MJ driver at 3.0 MV
varName    varValue    data type  comment
-----
IDStation  =  10        :integer    ! station identification #
Qsys       =  1.333333e-3 :real      ! system charge             [C]
N          =  16        :integer    ! number of beams           [ ]
Amu        =  200       :real      ! atomic no.                 [amu]
q          =  3         :real      ! charge state               [ ]
V          =  3         :real      ! cumulative voltage         [MV]
sig0       =  72.0      :real      ! undepressed tune           [deg]
epsn       =  1e-6      :real      ! nor. emittance, no pi     [m-r]
filename   =  'CostV1.010' :character ! name of file
-----

      There is one group of the above data for each of the stations in the
      system. The actual order of the data is immaterial; however, the values
      that follow the parameter IDStation are for that IDStation. Thus, each
      data group contains 8 values as shown above. The data consists of a
      variable name terminated by at least one trailing blank, then an equal
      sign followed by the value of the variable, followed by the data type
      represented as :<type>. The variable names are as shown above. The
      value of each data item is represented by a valid FORTRAN constant. In
      particular a 'text string' is used for the string text that names the
      file identifier of the file containing the associated Parameter values
      found when calculating the minimum cost.

      The module CalCost locates the data that corresponds to the given
      IDStation. It loads that data and then proceeds to calculate The
      minimum cost configuration for the given station.

      After this minimum cost configuration is found, the processes involved
      save the current state of the IDStation parameters, along with a
      fileName pointer that points to the files containing this data.

      The module CalCost is then exited and control is returned to the
      invoking module. Since the complete state of this minimum cost
      configuration is saved, it is possible to process it further. For
      example: reports may be generated, plots made, or smoothing techniques
      applied; or processes that minimize across the various stations with
      respect to given parameters and criteria may be developed and applied.

      The sole purpose of the module CalCost is to calculate, for the given
      station, the configuration that gives the minimum cost as constrained by
      the given constraints.

c  ENDFILE:  CalCost1.DAT

```

Figure 2-9. CostV1.DAT

```

c FILE: CostV1.DAT
c DATE: 910923

c EXAMPLE: dr4MJ@3MV

c Parameters that are assigned values from a data file:

c At the current station, HILDA cycles through the points
c in the parameter space defined below.

Beam Parameter Range Data for 4MJ driver at 3.0 MV
* varName      varValue      data type  comment
-----
c IDStation    10           :integer   ! station identification #

RLmin          = 0.35       :real      ! min. structure half-period    [m]
RLmax          = 0.35       :real      ! max. structure half-period    [m]
numRL          = 1          :real      ! # of grid points              [ ]

aMin           = 0.035      :real      ! min. beam size (max)          [m]
aMax           = 0.035      :real      ! max. beam size (max)          [m]
numa           = 1          :real      ! # of grid points              [ ]

delVmin        = 7.0E3      :real      ! min. voltage gain              [V]
delVmax        = 7.0E3      :real      ! max. voltage gain              [V]
numDelV        = 1          :real      ! # of grid points              [ ]

etaMin         = 0.40       :real      ! min. quad. packing fraction    [ ]
etaMax         = 0.40       :real      ! min. quad. packing fraction    [ ]
numEta         = 1          :real      ! # of grid points              [ ]
-----

c Set contents of the logfile CostV1.log using the value of iLog
iLog = 3 ! OK designs logged to terminal, skipped points not logged
      ! 1 OK and skipped designs logged to terminal
      ! 2 OK and skipped designs logged to CostV1.LOG
      ! 3 OK designs logged to CostV1.LOG
      ! 4 ONLY Minimum designs logged to CostV1.LOG
      ! 5 NO designs logged to CostV1.LOG or to the terminal

c ENDFILE: CostV1.DAT

```

Figure 2-10. AlSighAbar.DAT

```

c  FILE:  AlSighAbar.DAT
c  DATE:  900420

c  EXAMPLE:  dr4MJ@3MV

c  Parameters that are assigned values from a data file

*  The process AlSighAbar solves a transcendental equation to
*  find the value of a root alpha. We furnish here data that
*  is pertinent finding that solution. The user would usually have
*  NO need to change this data.

c  Parameters:
    c3      = 0.12      : ! eta coeff., see the process FAlpha

c  Used to find alpha:
*  The search for the root alpha is over this interval. This interval
*  should be sufficient. However, if it is not, then the process will
*  increase the interval.
    alphaMin = 1.0e-8    : ! min. alpha value
    alphaMax  = 5.0e00    : ! max. alpha value
    numdAlpha = 10       : ! num. of search intervals
    maxTry    = 2        : ! no. of search interval increases

c  Diagnostic printing
    iPrint    = 3        : ! print level, cumulative
                        : ! 0 no print out
                        : ! 1 accepted root print out
                        : ! 2 all found roots print
                        : ! 3 I/O variable print out
                        : ! n more print out/debug n .ge. 4

*  If a root cannot be found in the interval dalpha = (alphaMax - alphaMin)
*  then the search is redone in the interval starting at alphaMax and
*  ending at alphaMax + dalpha. This is done maxTry times.

c  Bounds on the depressed tune at the head of the beam:
    sigmaHMin = 0.0e00    : ! min. sigmaH      [deg]
    sigmaHMax = 90.0e00    : ! max. sigmaH      [deg]

c  ENDFILE:  AlSighAbar.DAT

```

Figure 2-11. TranMod.DAT

```

c FILE: TranMod.DAT
c DATE: 910923

c EXAMPLE: dr4MJ@3MV

c The module TranMod reads this data to select the
c element type and the design module that is used
c at the current station. This data set can be
c updated as more design modules are placed in HILDA.
  iType = 4      !! the element type
  iDesign = 30    !! the design module ScQ30

c The available selections are shown below:
c iType      Element Type
c   0          Drift space
c             iDesign  Design module
c             00        none
c   2          Bending magnet
c             iDesign  Design module
c             00        none
c   4          Quadrupoles Focusing/Defocussing
c             iDesign  Design module  Element Designed
c             10        EsQ10          Electrostatic, not yet in HILDA
c             20        FeQ20          Magnetic, Fe
c             30        ScQ30          Superconducting
c   6          Sextupole
c             iDesign  Design module
c             00        none

c ENDFILE: TranMod.DAT

```

Figure 2-12. FeQ20.DAT

```

c FILE: FeQ20.DAT
c DATE: 910923
c EXAMPLE: dr4MJ@3MV

c The process FeQ20 designs an FE magnetic that is basically like
c the ILSE FE magnet described in LBL PUB 5219.

c Parameters that are assigned values from a data file:

c Coefficients used in the Quadrupole Magnet Calculation:
cR1   = 1.25  !! aperture radius =  $R = cR1 * a + cR2$  [ ]
cR2   = 0.01  !! [m]
cdRp  = 0.03  !! pipe thickness      =  $delRPipe = cdRp * R$  [ ]
dRPMIn = 0.001 !! minimum pipe thickness [m]
cdRg  = 0.10  !! gap width          =  $delRGap = cdRg * R$  [ ]
dRGMin = 0.002 !! minimum gap width [m]
cdrWire = 0.005 !! wire layer width =  $delrWire$  [m]
c      !! =  $cdrWire * Bprime * rWire$  [m/T]
drWMin = 0.001 !! minimum wire width [m]
drFeMin = 0.002 !! minimum iron width [m]
czlOv  = 0.75  !! overhang length   =  $zlOver = czlOv * rWire$  [ ]
cPitch = 0.00  !! pitch              =  $2.0D00 rFe + cPitch$  [m]

c Limits
Bmax   = 1.5   !! pole tip field limit [T]
fZSpace = 0.10  !! non-magnet space limit, frac. of half-period [ ]
c      !!  $((fZSpace * RL + mag. length) .le. (RL))$ 
c      !! where RL = half-period [m]

```

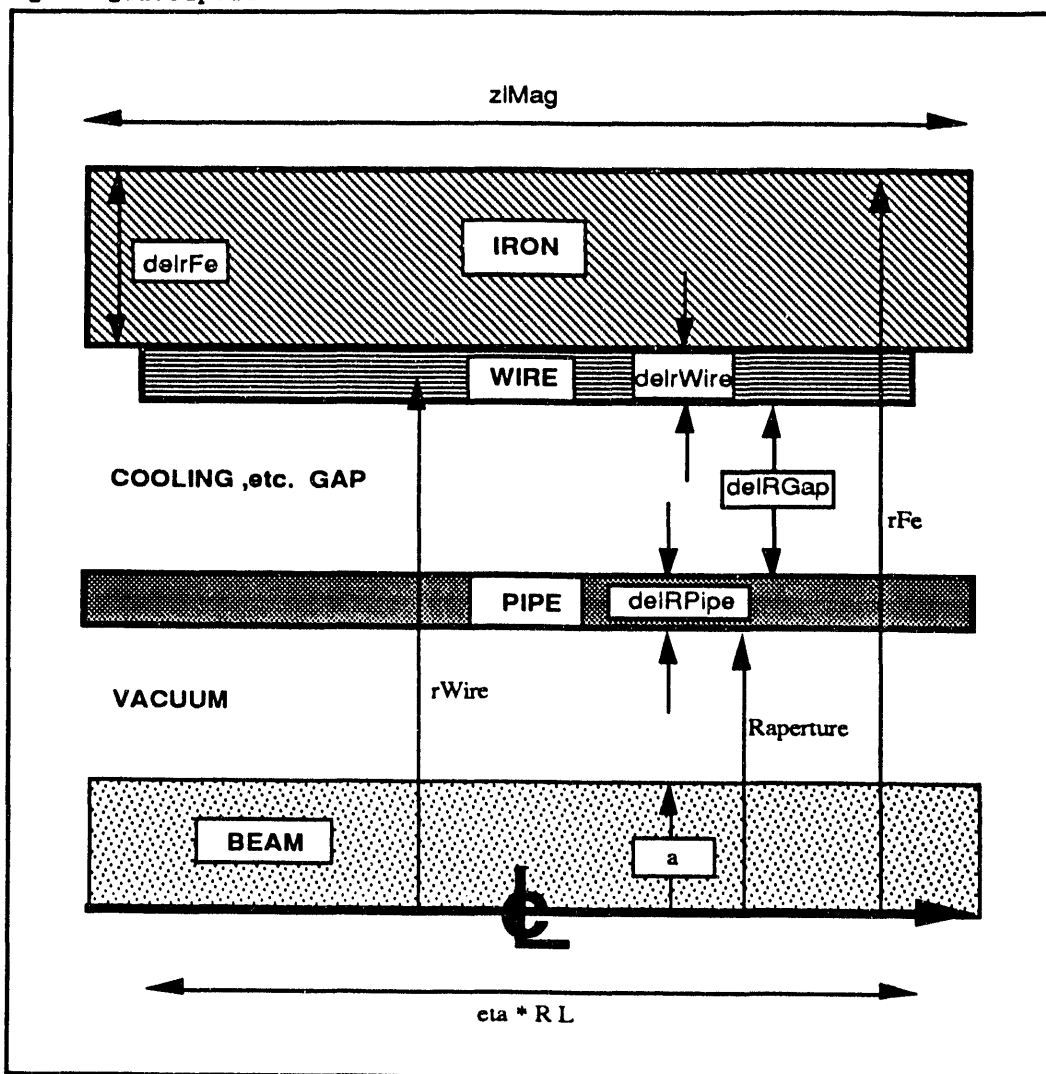

Figure 2-12. FeQ20.DAT (continued)

Material costs based on ILSE design, as furnished by C. Fong.
FeQ20 uses the weight of the Fe to get the total cost. The design
module FeQ20 uses this data for costing the quadrupole. There is only
one data set allowed for this version of the module.

```
denFe = 8.1657 !! yoke material density Low Carbon steel [g/cm**3]
cCost = 1.0    !! complexity factor                      [ ]
qCost = 1.0    !! quantity factor                       [ ]
uCost = 33.0   !! unit cost of the material              [$ /kg]
```

* Shown below is the FeQ20 quadrupole labeled with the variables
* used in FeQ20 and this associated data set.

FeQ20 Quadrupole



c ENDFILE: FeQ20.DAT

Figure 2-13. ScQ30.DAT

```

c FILE: ScQ30.DAT
c DATE: 910923

c EXAMPLE: dr4MJ@3MV

Parameters that are assigned values from a data file:
Derived from Bob Biere's SC Quadrupole Data in his Thesis.

This data set has only 1 data set for each of the below items.
Up to 10 data sets can be furnished for each item. The actual data
sets are furnished at the end of this ScQ30.DAT data file.

c Item                      Flag value    Material choice
c Quad SC Wire              1             NbTi  Niobium Titanium
c Non- SC Wire Space        1             Cu    Copper
c Quad Wrap (Stress)        1             Steel

c Quad Pipes                1             No Data on this Material
c Quad Insulation           1             No Data on this Material
c Quad Cooling layers       1             No Data on this Material
c Array outer Wrap          1             Steel

CHOOSE MATERIALS for the SC Quadrupole
iScCab = 1      !! material for the SC wires           [ ]
iCabS  = 1      !! material for the non-SC wire space  [ ]
iWrap  = 1      !! material for the Quad Wrap (stress) [ ]

iPipe  = 1      !! material for the vac. & outer Quad pipes [ ]
nPipe  = 2      !! # of pipes                           [ ]
drPipe = 0.0020 !! thickness of pipes                   [m]

iInsul = 1      !! material for the insulation layers
nInsul = 3      !! # of insulation layers               [ ]
drInsul = 0.0033 !! thickness of insulation layers      [m]

iCool  = 1      !! data set for the cooling layers
nCool  = 2      !! # of cooling layers                   [ ]
drCool = 0.0010 !! thickness of layers                  [m]

CHOOSE MATERIAL for the Array of numBeam Quadrupoles
iAWrap = 1      !! material for the outer wrap (collar)

SET LIMITS and PARAMETERS values
Quadrupoles
Limits on the Quadrupole SC Wire
drWmin = 1.0e-2 !! thickness, minimum                  [m]
drWmax = 0.10   !! thickness, maximum                  [m]
dShell = 0.0015 !! thickness of the outer shell        [m]
Non-Quadrupole free space
zQend  = 0.000  !! space used by end packaging for the quad [m]
          !! z1Mag = zQend + (eta*RL + z1Over)
f2Space = 0.10  !! non-magnet space limit, frac. of half-period [ ]
          !! (f2Space * RL + z1Mag) .LE. RL
          !! where RL = half-period [m]
Coefficients used in the Superconducting Quadrupole Calculation:
cR1    = 1.25   !! aperture radius = R = cR1 * a + cR2    [ ]
cR2    = 0.01   !!                                         [m]

```

Figure 2-13. ScQ30.DAT (continued)

```

Quadrupole Wrap (stress) scaling parameters
sWrap  = 0.01  !! wrap thickness used for BWrap, rWrap      [m]
BWrap  = 5.00  !! field used for scaling the Quad. Wrap    [T]
rWrap  = 0.12  !! radius used with BWrap                  [m]
Quadrupole SC Wire radius limit
rWtRL  = 0.50  !! rWire .lt. eta*RL * rWtRL                [ ]
Array of numBeam Quadrupoles
fCollar = 0.25  !! array wrap (collar) width, frac. of pitch [ ]

COST DATA
Quadrupole (Channel) Assembly-Complexity Cost factor
BFactor = 0.39  !! Multiplies the Quad (Wrap + Cable) cost [ ]
Array Assembly-Complexity Cost factor
cAFact  = 1.00  !! Multiplies the Array cost                [ ]

c  The Superconducting quadrupole that ScQ30 designs is shown below.
c  The labels are keyed to this data set and to the variables in the
c  design module.

*  The quadrupole that is designed is cylindrically symmetric about the
*  beam center line. The pitch is the distance from beam-center to beam-
*  center. The individual quadrupoles are then stacked as an array to make
*  a quadrupole package, or bundle. The stacking is done here and the outer
*  radius of the bundle is returned for later use.
*
*  In Figure 1 is shown a side view of the ScQ30 quadrupole, labeled with
*  the variables used in ScQ30. In Figure 2 an end view of the same
*  quadrupole is shown. In Figure 3 an end view is shown for a four beam
*  array.
*
*  The end plates for the SC quadrupole have a nominal length of rWo.
*  However, that longitudinal length can be changed by supplying a non-zero
*  value for the parameter zQend. The value of zQend can be positive or
*  negative. The free space that is left at the ends of the quadrupole, and
*  hence at the ends of the array package, is determined by the value of
*  the parameter fzspace. Both of these parameters obtain their values from
*  data on the file ScQ30.DAT. The length of the quadrupole is determined
*  by eta, the packing factor, and by RL, the lattice half-period. These
*  quantities are the current values supplied when the module ScQ30.DAT is
*  invoked. All volumes, and thus all weights and material costs, use zLMag
*  as the longitudinal length. This is not exactly right, however that is
*  the way this version of ScQ30 does the calculation.
*
*  The end-view picture shown in Figure 3 has three insulation layers, two
*  cooling layers, an inner vacuum pipe and an outer enclosing pipe. The
*  windings are outside the outer pipe, as shown. This assembly is enclosed
*  in an outer wrap which itself is covered by a shell. This figure should
*  be referred to the data in the file ScQ30.DAT. We note that the amount
*  of conductor that is required is calculated as though it filled the
*  annular ring of thickness drWire, as shown in Figure 2. Although this is
*  not quite right, it is the way this version of ScQ30 does the
*  calculation.
*
*  The individual quadrupoles are placed together in a square package as
*  shown in Figure 3. In order that this be possible, it is necessary that
*  the number of beams be a perfect square, such as 4, 9, 16, 25, etc.
*  ScQ30 checks this condition and completes the quadrupole array design
*  only if this is the case.

```

Figure 2-13. ScQ30.DAT (continued)

- * The outer wrap containing the quadrupole array has a thickness that is
- * determined by the parameter $fCollar$. This parameter obtains its value
- * from the data file ScQ30.DAT. The enclosing circle for the whole
- * quadrupole array assembly, including the outer wrap, has a radius
- * $roTArray$. Any elements that contain the focussing array must have in
- * inner radius of at least this value. In particular, the acceleration
- * cores must have an inner bore of this radius, or more, when the
- * quadrupole array is allowed to penetrate into the cores.

ScQ30 Quadrupole Side View

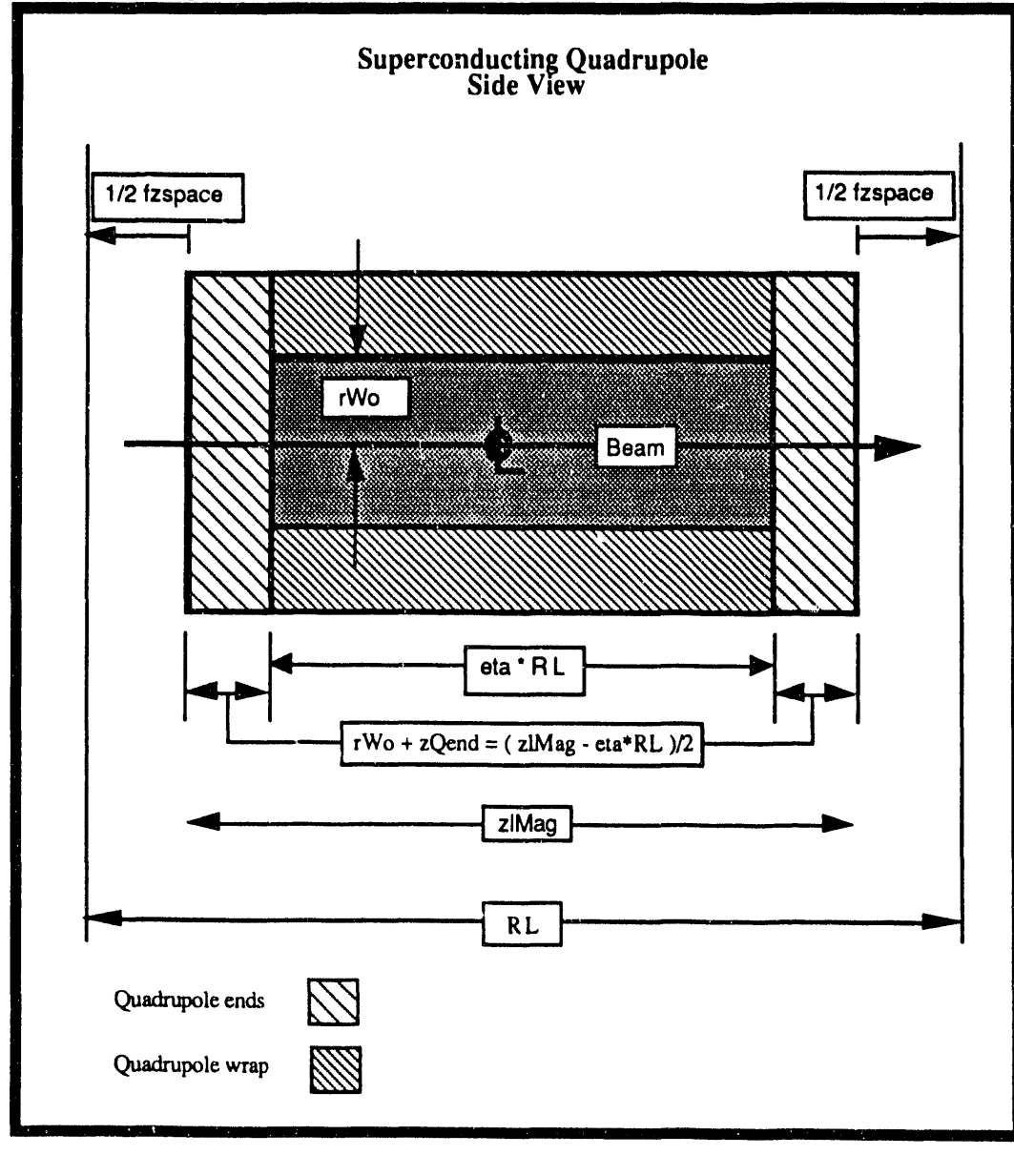


Figure 2-13. ScQ30.DAT (continued)

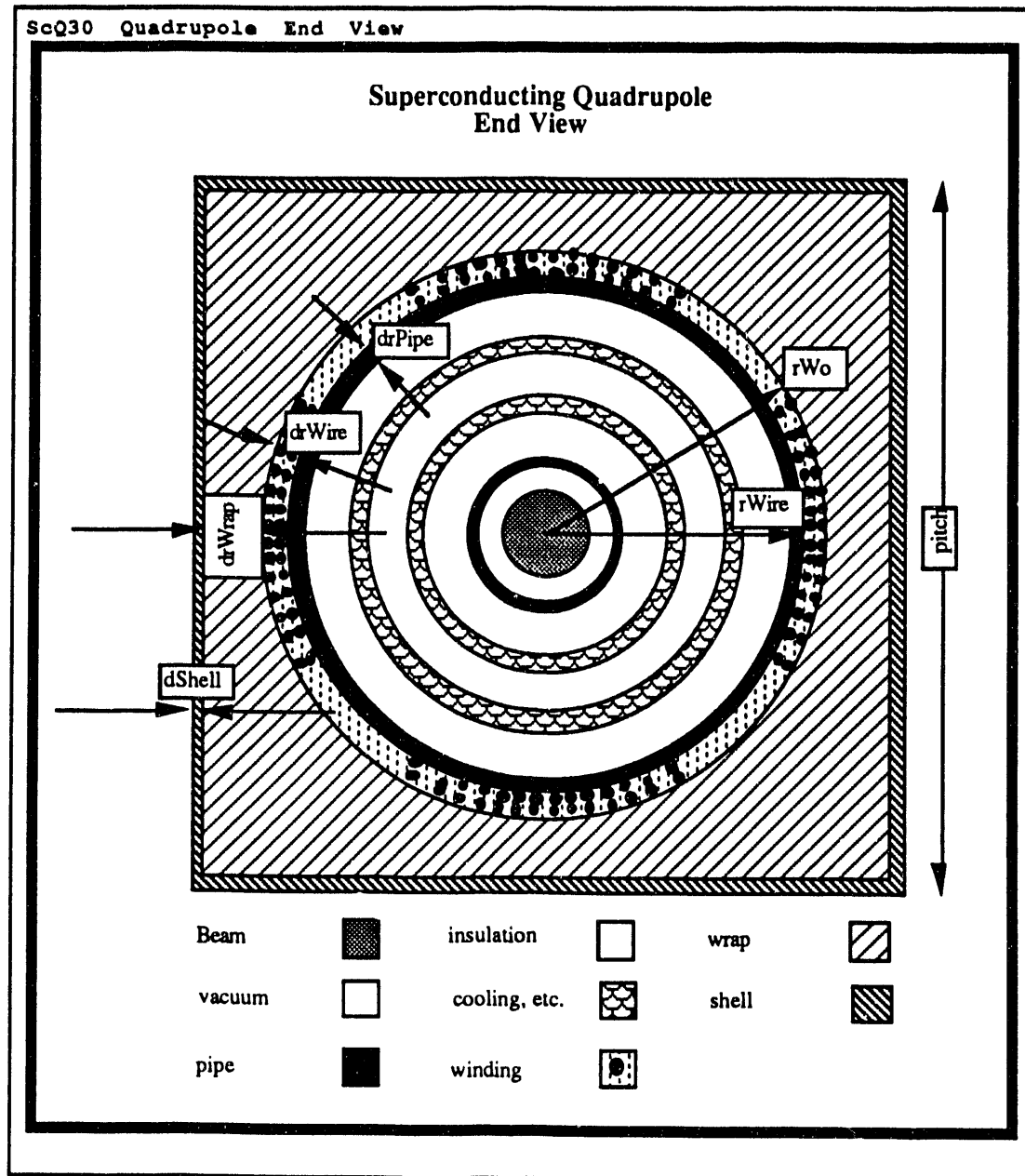


Figure 2-13. ScQ30.DAT (continued)

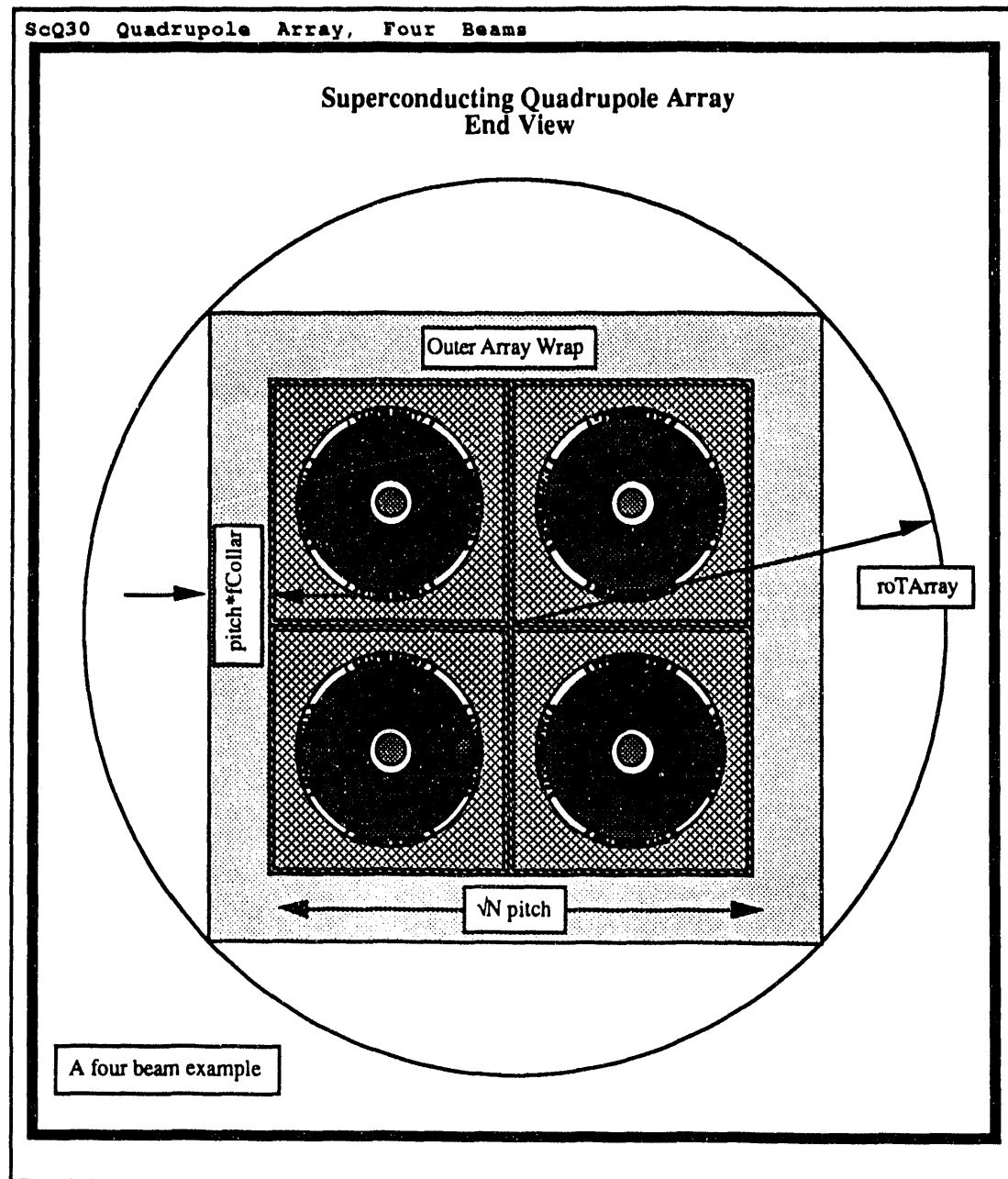


Figure 2-13. ScQ30.DAT (continued)

MATERIAL DATA and COSTS are furnished below.
Each item can have up to 10 material data sets

Superconducting Cable

```

nScMat      = 1          !! # of material data sets
IDScMat (n) =            'NbTi Niobium Titanium'
denScMat (n) = 7.6e3      !! material density          [kg/m**3]
uCScMat (n) = 300.00     !! unit cost                  [$ /kg]
qFScMat (n) = 1.0        !! quantity factor            [ ]
cFScMat (n) = 1.0        !! complexity factor          [ ]
Current density parameters and Field limits
cJCcoeff (n) = 2.9e9     !! slope of cJ curve          [A/m**2]
cJBn (n)    = 10.0       !! numerator field parameter  [T]
cJBd (n)    = 5.0        !! denominator field parameter [T]
rLamda (n)  = 0.3571     !! SC wire packing fraction   [T]
BWomax (n)  = 10.0       !! B at outer wire, maximum   [T]

```

Non SC Cable

```

nCMat      = 1          !! # of material data sets
IDScMat (n) =            'Cu Copper'
denCMat (n) = 8.9e3      !! material density          [kg/m**3]
uCCMat (n) = 50.0        !! unit cost                  [$ /kg]
qFCMat (n) = 1.0        !! quantity factor            [ ]
cFCMat (n) = 1.0        !! complexity factor          [ ]

```

Quadrupole Wrap (Stress) Material

```

nWMat      = 1          !! # of material data sets
IDScMat (n) =            'Steel'
denWMat (n) = 8.1657e3   !! material density          [kg/m**3]
uCWMat (n) = 25.0        !! unit cost                  [$ /kg]
qFWMat (n) = 1.0        !! quantity factor            [ ]
cFWMat (n) = 1.0        !! complexity factor          [ ]

```

Array Outer Wrap (Collar) Material

```

nAOtMat     = 1          !! # of material data sets
IDScMat (n) =            'Steel'
denAOtMat (n) = 8.1657e3 !! material density          [kg/m**3]
UCAOtMat (n) = 25.0      !! unit cost                  [$ /kg]
qFAOtMat (n) = 1.0      !! quantity factor            [ ]
cFAOtMat (n) = 1.0      !! complexity factor          [ ]

```

Pipe Material, Vacuum and around Quad outer insulation layer

```

nPMat      = 1          !! # of material data sets
IDScMat (n) =            'No Data on this Material'
denPMat (n) = 0.0        !! material density          [kg/m**3]
uCPMat (n) = 1.0        !! unit cost                  [$ /kg]
qFPMat (n) = 1.0        !! quantity factor            [ ]
cFPMat (n) = 1.0        !! complexity factor          [ ]

```

Insulation (thermal) Material

```

nIMat      = 1          !! # of material data sets
IDScMat (n) =            'No Data on this Material'
denIMat (n) = 0.0        !! material density          [kg/m**3]
uCIMat (n) = 1.0        !! unit cost                  [$ /kg]
qFIMat (n) = 1.0        !! quantity factor            [ ]
cFIMat (n) = 1.0        !! complexity factor          [ ]

```

Figure 2-13, ScQ30.DAT (continued)

```

Cooling Layers, data sets
nClay      = 1          :! # of material data sets
IDScMat (n) =          'No Data on this Material'
denClay(n) = 0.0        :! material density          [kg/m**3]
uCClax (n) = 1.0        :! unit cost                  [$ /kg]
qFCLay (n) = 1.0        :! quantity factor           [ ]
cFCLay (n) = 1.0        :! complexity factor          [ ]

c  ENDFILE:  ScQ30.DAT

```

Figure 2-14. StrucCore.DAT

```

c  FILE:  StrucCore.DAT
c  Date:  910923
c  EXAMPLE:  dr4MJ@3MV

c  Parameters that are assigned values from a data file:
c  This data set has been used for the 4MJ Driver Example.

c  SET ERROR LOGGING TO TERMINAL
  iLog      = 0      : ! 0 do not log failed designs to terminal
              : ! 1      log failed designs to terminal

c  SELECT COMPONENT MATERIALS for the acceleration module
c  Uses data in the data sets below:

c  CORE amorphous tape materials available
c  1 Metglas 2605 S2 - wound and annealed:
c  2 Metglas 2605 C0 - wound and annealed:
  idAm      = 1      : ! tape used

c  CORE amorphous tape widths available
c  1 5.08 [cm]
c  2 10.16 [cm]
c  3 17.018 [cm]
c  4 20.32 [cm]
  idWTape   = 4: ! tape width used

c  CORE/CELL HOUSING materials available
c  1 Low Carbon steel - simple machined - 1020
c  2 Low Carbon steel - welded & machined - 1020
c  3 Aluminum          - simple machined 6061
c  4 Aluminum          - welded & machined 6061
c  5 Stainless steel   - simple machined - 304
c  6 Stainless steel   - welded & machined - 304
c  7 Stainless steel   - simple machined - 316
c  8 Stainless steel   - welded & machined - 316
c  9 Aluminum Casting - sand - raw 356
c  10 Aluminum Casting - sand - 356 w/ simple machining

  idCSM     = 2      : ! core power lead, outer housing, inner bobbin
  idCH      = 2      : ! cell housing

```


Figure 2-14. StrucCore.DAT (continued)

```

c  INSULATION/DIELECTRIC  materials
c  1  NEMA g-10 Composite - machined
c  2  Polyethylene LP-390-C Dielectric, injection molded
c  3  Water deionized
c  4  Freon
c  5  SF6
c  6  Vacuum

    idIN      = 2 : ! core/cell housing insulation
    idDiC     = 4 : ! dielectric coolant
    idGP      = 6 : ! gap operating pressure range

c  GAP INSULATOR MATERIAL.
c  1 Alumina - pressure cast & brazed large dia/ to 58" by 1" thick

    idGI      = 1 : ! gap insulator

c  PULSE FORMING NETWORK,    available PFN peak kilovolts per core
    pV        = 200.0 : ! 80 - 500      [kV]
*                                     The module StrucCore will use this to
*                                     determine how many modules are needed to
*                                     furnish the required voltage gain.

c  MINIMUM LENGTH LEFT AVAILABLE FOR quad. support,etc.
    QAl       = 10.0 : ! z-axis          [% of half-period length]

c  MAXIMUM LENGTH AVAILABLE FOR gap insulator.
    gILmax    = 80.0 : ! z-axis          [% of cell length cellL ]

c  HOUSING THICKNESSES:
c  station    ! ave. of data for weights of 5000lb & 11000lb
    z1H      = 2.0 : ! cell housing thickness, 5000lb/11000lb [cm]
    z1PH     = 1.0 : ! core housing/power lead thickness      [cm]
    proC     = 2.0 : ! top of core      delTC = proC * z1PH
    priM     = 2.0 : ! bottom of core   delBC = priM * z1Ph
    proH     = 2.0 : ! cell housing     delCH = proH * z1H

* The accelerations structure that is designed is shown in
* Figure 1 Acceleration Structure and Figure 2 Acceleration Module
* below. The labels used in those figures correspond to the identifiers
* used in this data set. These identifiers are the same as used in
* the design's module StrucCore that uses this data.

```

Figure 2-14. StrucCore.DAT (continued)

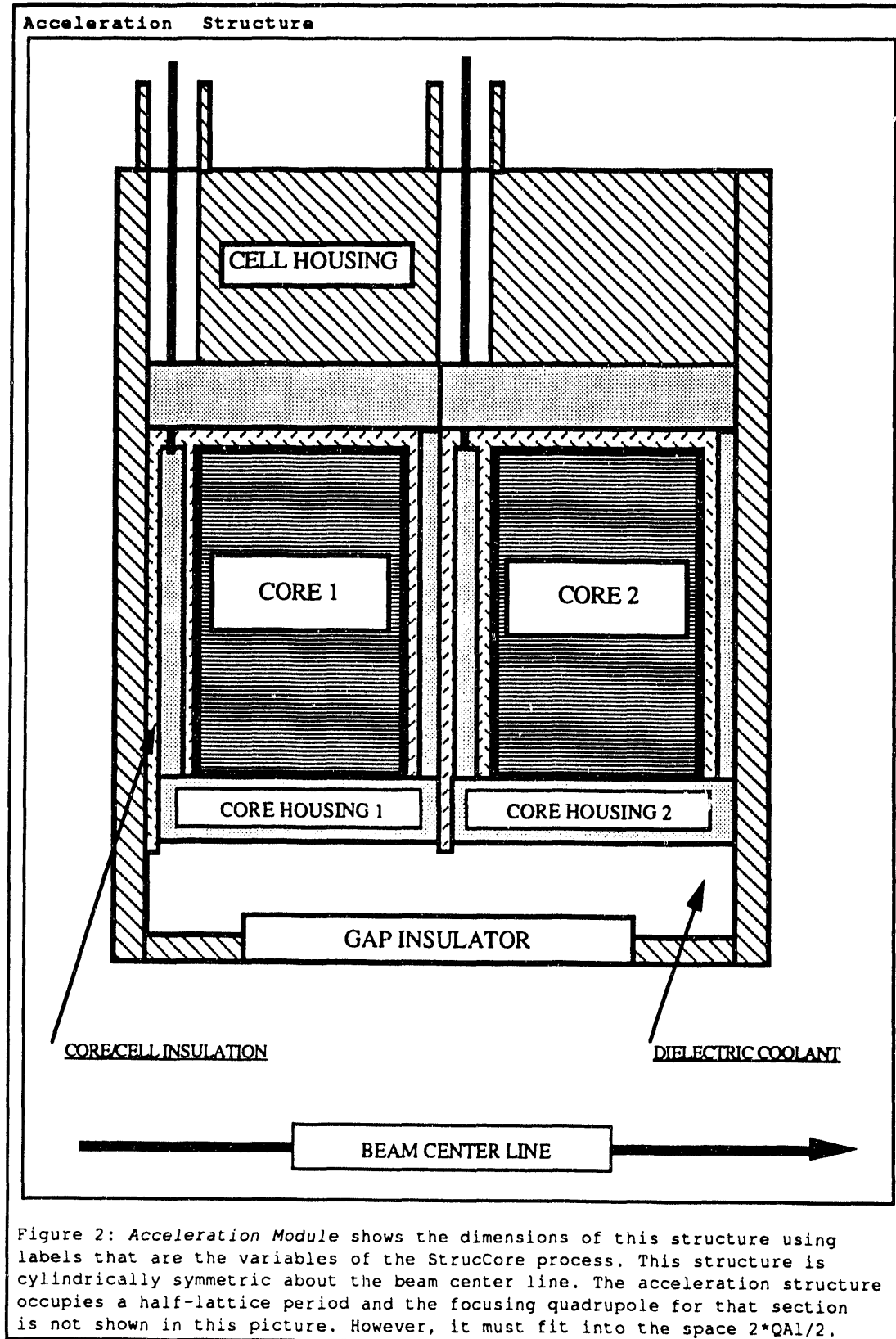


Figure 2: Acceleration Module shows the dimensions of this structure using labels that are the variables of the StrucCore process. This structure is cylindrically symmetric about the beam center line. The acceleration structure occupies a half-lattice period and the focusing quadrupole for that section is not shown in this picture. However, it must fit into the space $2 \cdot Q_{A1}/2$.

Figure 2-14. StrucCore.DAT (continued)

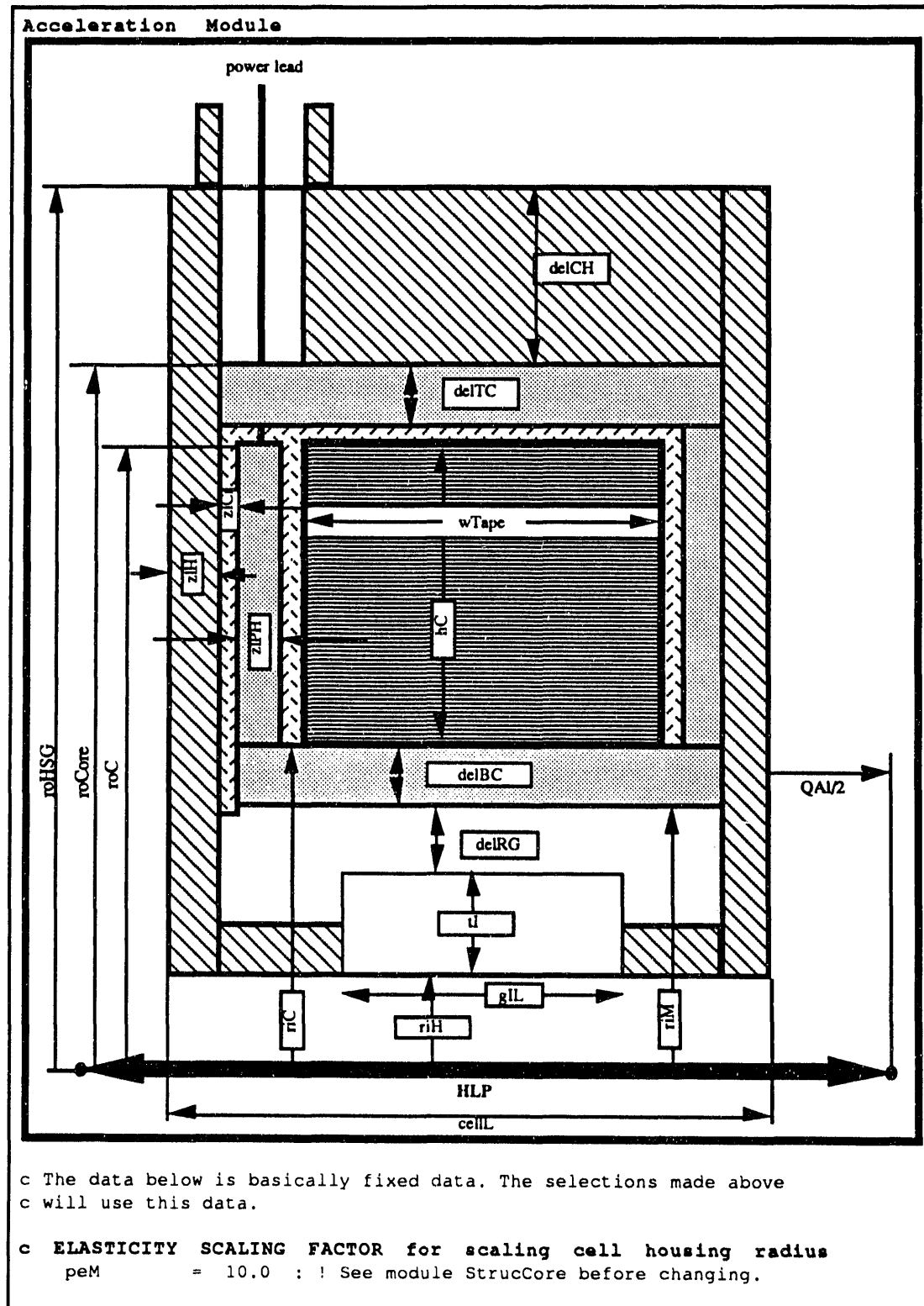


Figure 2-14. StrucCore.DAT (continued)

```

c  NUMBER OF BEAMS DATA:
  nBSets      = 4      : ! # of sets of beam data, .le. maxBSets

c    Beam configurations
  nBeams(1)   = 4
  nBeams(2)   = 16
  nBeams(3)   = 21
  nBeams(4)   = 64

c    4 Beam data:
  ti(1)       = 2.5    : ! acc. gap insulator thickness      [cm]
  fQ(1)       = 1.1    : ! quad array perimeter support factor [ ]

c    16 Beam data:
  ti(2)       = 3      : ! acc. gap insulator thickness      [cm]
  fQ(2)       = 1.1    : ! quad array perimeter support factor [ ]

c    21 Beam data:
  ti(3)       = 2      : ! acc. gap insulator thickness      [cm]
  fQ(3)       = 1.05   : ! quad array perimeter support factor [ ]

c    64 Beam data:
  ti(4)       = 4.0    : ! acc. gap insulator thickness      [cm]
  fQ(4)       = 1.05   : ! quad array perimeter support factor [ ]

c  MATERIAL PROPERTIES
c  For each material furnish a list of properties

c  AMORPHOUS CORE TAPES:
  nAmSets      = 2      : ! number of material sets, .le. 10 sets

c  1  Metglas 2605 S2 - wound and annealed:
  denAm (n)    = 7.1800 : ! density                          [g/cm**3]
  cFacAm (n)   = 1      : ! complexity factor                [ ]
  qFacAm (n)   = 1      : ! quantity factor                  [ ]
  uCostAm(n)   = 5.0    : ! unit cost                         [$ /kg]
  pFam (n)     = 0.80   : ! radial packing factor            [ ]
  delBAm (n)   = 2.5    : ! flux swing                       [T]

c  1  Metglas 2605 S2 - wound and annealed:
  denAm (n)    = 7.5600 : ! density                          [g/cm**3]
  cFacAm (n)   = 1      : ! complexity factor                [ ]
  qFacAm (n)   = 1      : ! quantity factor                  [ ]
  uCostAm(n)   = 40.0   : ! unit cost                         [$ /kg]
  pFam (n)     = 0.725  : ! radial packing factor            [ ]
  delBAm (n)   = 2.5    : ! flux swing                       [T]

*  The cores are sized in StrucCore using the above value of the
*  flux swing along with the required voltage gain.

c  AMORPHOUS CORE TAPE WIDTHS and corresponding max. aspect ratios:
  nWRSets      = 4      : ! number of width/ratio sets, .le. 10 sets

  wTape (1)    = 5.08   : ! width                            [cm]
  aRCmax(1)    = 10.0   : ! max. height to width aspect ratio [ ]

  wTape (2)    = 10.16  : ! width                            [cm]
  aRCmax(2)    = 8.0    : ! max. height to width aspect ratio [ ]

  wTape (3)    = 17.018 : ! width                            [cm]
  aRCmax(3)    = 6.00   : ! max. height to width aspect ratio [ ]

  wTape (4)    = 20.32  : ! width                            [cm]
  aRCmax(4)    = 4.0    : ! max. height to width aspect ratio [ ]

```

Figure 2-14. StrucCore.DAT (continued)

```

c  CORE MODULES and CELL HOUSING materials that are available.
    numMat      = 10      : ! number of materials in this set .le. 20

c  1  Low Carbon steel - simple machined - 1020
    denMat      (n) = 8.1657 : ! density [gm/cm**3]
    cFacMat     (n) = 1      : ! complexity factor {}
    qFacMat     (n) = 1      : ! quantity factor {}
    uCostMat    (n) = 9.8987 : ! unit cost [$ /kg]
    emMat       (n) = 30     : ! [10**6 lb/in**2]

c  2  Low Carbon steel - welded & machined - 1020
    denMat      (n) = 8.1657 : ! density [gm/cm**3]
    cFacMat     (n) = 1      : ! complexity factor {}
    qFacMat     (n) = 1      : ! quantity factor {}
    uCostMat    (n) = 9.8987 : ! unit cost [$ /kg]
    emMat       (n) = 30     : ! [10**6 lb/in**2]

c  3  Aluminum -simple machined 6061
    denMat      (n) = 2.7123 : ! density [gm/cm**3]
    cFacMat     (n) = 1      : ! complexity factor {}
    qFacMat     (n) = 1      : ! quantity factor {}
    uCostMat    (n) = 13.8449 : ! unit cost [$ /kg]
    emMat       (n) = 10     : ! [10**6 lb/in**2]

c  4  Aluminum - welded & machined 6061
    denMat      (n) = 2.7123 : ! density [gm/cm**3]
    cFacMat     (n) = 1      : ! complexity factor {}
    qFacMat     (n) = 1      : ! quantity factor {}
    uCostMat    (n) = 17.1959 : ! unit cost [$ /kg]
    emMat       (n) = 10     : ! [10**6 lb/in**2]

c  5  Stainless steel - simple machined - 304
    denMat      (n) = 7.9470 : ! density [gm/cm**3]
    cFacMat     (n) = 1      : ! complexity factor {}
    qFacMat     (n) = 1      : ! quantity factor {}
    uCostMat    (n) = 14.5283 : ! unit cost [$ /kg]
    emMat       (n) = 29     : ! [10**6 lb/in**2]

c  6  Stainless steel - welded & machined - 304
    denMat      (n) = 7.9470 : ! density [gm/cm**3]
    cFacMat     (n) = 1      : ! complexity factor {}
    qFacMat     (n) = 1      : ! quantity factor {}
    uCostMat    (n) = 17.8790 : ! unit cost [$ /kg]
    emMat       (n) = 29     : ! [10**6 lb/in**2]

c  7  Stainless steel - simple machined - 316
    denMat      (n) = 7.9470 : ! density [gm/cm**3]
    cFacMat     (n) = 1      : ! complexity factor {}
    qFacMat     (n) = 1      : ! quantity factor {}
    uCostMat    (n) = 15.9834 : ! unit cost [$ /kg]
    emMat       (n) = 29     : ! [10**6 lb/in**2]

c  8  Stainless steel - welded & machined - 316
    denMat      (n) = 7.9470 : ! density [gm/cm**3]
    cFacMat     (n) = 1      : ! complexity factor {}
    qFacMat     (n) = 1      : ! quantity factor {}
    uCostMat    (n) = 19.3343 : ! unit cost [$ /kg]
    emMat       (n) = 29     : ! [10**6 lb/in**2]

c  9  Aluminum Casting - sand - raw 356
    denMat      (n) = 2.7123 : ! density [gm/cm**3]
    cFacMat     (n) = 1      : ! complexity factor {}
    qFacMat     (n) = 1      : ! quantity factor {}
    uCostMat    (n) = 9.9427 : ! unit cost [$ /kg]
    emMat       (n) = 10     : ! [10**6 lb/in**2]

```

Figure 2-14. StrucCore.DAT (continued)

```

c 10 Aluminum Casting - sand - 356 w/ simple machining
    denMat      (n) = 2.7123 : ! density [gm/cm**3]
    cFacMat      (n) = 1 : ! complexity factor []
    qFacMat      (n) = 1 : ! quantity factor []
    uCostMat     (n) = 13.4481 : ! unit cost [$ /kg]
    emMat        (n) = 10 : ! [10**6 lb/in**2]

c STRUCTURE INSULATION/DIELECTRIC materials
    numInMat     = 6 : ! number of materials in this set .le. 20

c 1 NEMA g-10 Composite - machined
    denInMat     (n) = 1.9222 : ! density [gm/cm**3]
    cFacInMat     (n) = 1 : ! complexity factor []
    qFacInMat     (n) = 1 : ! quantity factor []
    uCostInMat    (n) = 27.1827 : ! unit cost [$ /kg]
    VoltInMat     (n) = 197.00 : ! voltage breakdown [kV/cm]

c 2 Polyethylene LP-390-C Dielectric, injection molded
    denInMat     (n) = 0.9850 : ! density [gm/cm**3]
    cFacInMat     (n) = 1 : ! complexity factor []
    qFacInMat     (n) = 1 : ! quantity factor []
    uCostInMat    (n) = 20.6417 : ! unit cost [$ /kg]
    VoltInMat     (n) = 197.00 : ! voltage breakdown [kV/cm]

c 3 Water deionized
    denInMat     (n) = 1.0000 : ! density [gm/cm**3]
    cFacInMat     (n) = 1 : ! complexity factor []
    qFacInMat     (n) = 1 : ! quantity factor []
    uCostInMat    (n) = 0.0 : ! unit cost [$ /kg]
    VoltInMat     (n) = 1.0 : ! voltage breakdown [kV/cm]

c 4 Freon
    denInMat     (n) = 1.8 : ! density [gm/cm**3]
    cFacInMat     (n) = 1 : ! complexity factor []
    qFacInMat     (n) = 1 : ! quantity factor []
    uCostInMat    (n) = 1.66 : ! unit cost [$ /kg]
    VoltInMat     (n) = 39.00 : ! voltage breakdown [kV/cm]

c 5 SF6
    denInMat     (n) = 1.0 : ! density [gm/cm**3]
    cFacInMat     (n) = 1 : ! complexity factor []
    qFacInMat     (n) = 1 : ! quantity factor []
    uCostInMat    (n) = 0.0 : ! unit cost [$ /kg]
    VoltInMat     (n) = 8.00 : ! voltage breakdown [kV/cm]

c 6 Vacuum
    denInMat     (n) = 0.0000 : ! density [gm/cm**3]
    cFacInMat     (n) = 1 : ! complexity factor []
    qFacInMat     (n) = 1 : ! quantity factor []
    uCostInMat    (n) = 0.0 : ! unit cost [$ /kg]
    VoltInMat     (n) = 50.000 : ! voltage breakdown [kV/cm]

c GAP INSULATOR materials
    numGI        = 1 : ! number of materials in this set .le. 10

c 1 Alumina - pressure cast & brazed large dia. to 58" by 1" thick
    denGapI      (1) = 3.7170 : ! density [gm/cm**3]
    cFacGapI      (1) = 1 : ! complexity factor []
    qFacGapI      (1) = 1 : ! quantity factor []
    uCostGapI     (1) = 428.2876 : ! unit cost [$ /kg]
    VoltGapI      (1) = 12.0 : ! voltage breakdown [kV/cm]

c END: StrucCore.DAT

```

Analyzing the Output and Generating Reports

The basic output generated during a HILDA run is the file CostV1.NNN. In this example this is CostV1.010, since the calculation of the minimum-cost design was at station 10. In the discussion below we refer to this particular file with the understanding that it represents any of the CostV1.NNN files that HILDA generates.

There is an associated logfile, CostV1.LOG, which in this 3MV example does not contain any information that we want to save. Therefore, the file was not saved. If we had scanned over a parameter space with many points, this file could have been used to save information about the rejected designs and information about the designs that were completed but were not the minimum in cost.

The present version of HILDA does not process the CostV1.010 file. We have analyzed the results for the cases that we have run by downloading that file into a PC and then using a spreadsheet to analyze the results and create reports. The fact that these tools are available and well developed should be strongly considered before time is spent in creating HILDA modules that can do these analysis tasks. Future versions of HILDA will contain modules that enable the user to easily process the CostV1.010 and create a file that is directly readable by a spreadsheet, or graphing program. We have presently used a separate program that converts the file to a tab delimited file; this being a form that most spreadsheet, or graphing, programs will read. It is strongly recommended that the basic logfile NOT contain tab characters. It has been our experience that tabs can cause many problems when files are read by different programs in different environments. The HILDA convention has been NO tabs, unless there is a specific reason for needing them.

In the present example there really is not much to analyze. What we have is a specific design for a particular station. We could use this output to generate drawings for the design. We could also compare this solution with nearby solutions, by rerunning HILDA. In the 4MJ driver example, from which this example is taken, we have seven stations of design output. In that case we can use these files to estimate a total cost of the complete machine. The complete 4MJ driver example is on the HPD disks.

Example 2: The 3000MV Station of a 4MJ Driver

We point out first that the example described here is in the folder *dr4MJ @ 3000MV*, which is in the folder *MSW/Hilda/DAT* of the HPD disks. The *ReadMe* files in those folders contain additional information about this example. The information that we present here has been extracted from files in those folders. This 3000MV example and all its output can be reconstructed from the data files in *dr4MJ @ 3000MV*. We also note that the 3000MV example set up when HILDA is installed following the instructions in the HPD section *Maintaining HILDA: Installation* is precisely the example that we are describing here

Running HILDA at the 3000MV station is little different from running HILDA at the 3MV station. The difference is that the data must be correct for the 3000MV station. Many of the data files are the same at both stations. As we have previously noted the beam definition file *CalCost1.DAT* can contain more than one station. If the *CalCost1.DAT* data file for the 3MV example had also contained a data packet for the 3000MV station, then that file could have been used for both of these examples. It turns out that the basic data difference for these two examples is the range over which the HILDA parameters are varied. The other module data files are the same.

The output is, however, different. The program designs a superconducting quadrupole that has parameters that are appropriate for focusing a high energy beam and the acceleration cell must provide not only space for this quadrupole, but also must provide a 2.8MV energy gain.

We shall assume in this example that the 3MV example has been read and understood. Thus, we will not need the detailed explanations that were given in that example.

Running HILDA on the VAX

In the discussion that follows we assume that:

- HILDA has been installed by following the instructions in section *Maintaining Hilda: Installation*
- you are logged onto your VAX account and that the directory is [USER.HILDA].
- you have executed the LOGIN.COM file that was loaded into this directory
- the installed executable image HILDA.EXE exists in the directory [USER.HILDA.EXE]
- the 3000MV data files were installed in the directory [USER.HILDA.DAT.DAT3000MV]

All the above assumptions will be true if you have installed HILDA as recommended.

If these assumptions are true, then follow the steps below.

- Transfer to the execution directory by typing
EXE
- Check that you are in the [USER.HILDA.EXE] directory by typing
SHOW DEFAULT
If you are not in this directory something is wrong.
- Set up the HILDA module data files by typing
SET3000MV
At this point the HILDA module data files:
CalCost1.DAT
AlSighBar.DAT
CostV1.DAT
TranMod.DAT
ScQ30.DAT
StrucCore.DAT
contain the 3000MV data of this example.

- Run HILDA by typing
`RUNHILDA`
The main program HILDA will execute and you will be prompted for input. Follow the prompts.
- Continue with the example by entering in lower case the command
`cost`
You will be asked at what station to calculate the cost. The station number that we have used in this example for the 3000MV data is station 70.
- Type in the integer
`70`
As HILDA proceeds to calculate the minimum-cost design for the selected station it writes messages to the terminal to indicate what is happening. When it is finished with this design task it will save the results of the design on the file `CostV1.070`.
In this particular example the data for the parameter search grid has been set for the 3000MV station. You can if you wish, ask HILDA to calculate the cost of a design for one of the other stations that are in the data set `CalCost1.DAT`. However, we will instead:
- Exit from HILDA by typing the lower case command
`stop`
- Confirm this desire to stop by typing the upper case command
`YES`

This completes the actual running of HILDA.

If you did calculate the cost at another station, say station 60, remember that the present version of HILDA would have used the current data in the module data files. There is no way to update this data from within HILDA, for this version. The beam data would be as defined in the data file `CalCost1.DAT` and thus it would be correct for station 60, since data for that station is included in the file. However, the search grid might not be what you wanted to use.

A copy of the terminal output from HILDA for this example is on the HPD disks that are associated with this report. It is in the *HILDA/Terminal/3000MV* document that is in the *HILDA/Log* folder of this example. You can find the *HILDA/Log* folder in the *dr4MJ @ 3000MV* folder; see the Appendix section *Guide to the HPD Disks* of this report.

The Output File `CostV1.070`

The basic output file generated during this HILDA run has the name `CostV1.070`, since we have produced a design at station 70. Had we requested designs at other stations, we would also have those files as HILDA output files from this run.

The `CostV1.070` file generated by this example contains much information. In what follows below we present short selections from the file `CostV1.070`. The complete output file for this example can be found in the folder *HILDA.LOG*, which is in the *dr4MJ @ 3000MV* folder. The interested reader can find these folders in the HPD disks. We also note that this file is essentially the same file as was generated for the 3MV case; i.e., `CostV1.010`. A fuller description of the output file `CostV1.NNN` is given there.

HILDA Output File CostV1.070

The primary results of the HILDA calculation are at the beginning of the CostV1.070 file and are shown in Figure 2-15, *HILDA Output A*.

This output shows the values of the HILDA parameters that give this minimum-cost design. Following this is the total cost for the station components and the how much it costs to furnish a volt in energy gain. This output is the primary output from the HILDA calculation at the station 70.

Next come the input variables that define the beam at this 3000MV station. These are the system charge Q_{sys} , the choice of mercury ions with a charge state of 3, the undepressed tune (single beam, no space charge) of 72 degrees, and the normalized emittance of $.000001 \pi$ meter-radian.

In Figure 2-16, *HILDA Output B* we show secondary output from the HILDA run at station 70. This consists of: the number of Coulombs in each of the 16 beams, the beam pulse-length times the beam current, the dimensionless dynamic quantities for the mercury ions, followed by the beam magnetic rigidity.

The parameter grid that HILDA scanned over is then shown. For this example we have only one point. This point has been set to the parameter values that give the minimum-cost design at the 3000MV station. This has been taken from the 4MJ example that is in the HPD disks.

The average times for the calculation include reading the data file associated with each of the modules; also any other setup tasks required for the calculation. If there were many points in the grid we should find that the average times reported were less per grid point.

The Figure 2-17, *HILDA Output C* contains intermediate results. The quantity *alpha* is a parameter relating to the beam dynamics. The parameter *sigmaH* is the depressed tune at the head of the beam, in each of the 16 beams; *abar* is an average beam size. We record the perveance of the beam, the current, the pulse width (longitudinal length), the average voltage gain per meter (electric field strength), and the volt-seconds furnished by the acceleration cell. These items are explained in the section *The HILDA Model: Beam Transport Equations* of this report.

In Figure 2-18, *HILDA Output D* is shown that section of the output file CostV1.OUT that has been copied from the output file created by the module ScQ30. This contains all the design parameter information for the superconducting quadrupole associated with the minimum-cost design. We should note that this is not necessarily the minimum-cost quadrupole array. The minimum-cost design is arrived at by calculating the total cost at the station, in this case the 3000MV station of the 4MJ driver. Thus, HILDA will have also included the acceleration cell cost when selecting the minimum-cost design.

The variable *dolTArray* is the total dollar cost of the quadrupole array; in this case a 16 beam array. The variable *roTArray* is the outer radius of the array. The acceleration cell designed by the module StrucCore uses this quantity to determine the core inner diameter needed to accommodate this quadrupole array.

The rest of the information in this ScQ30 output pertains to the design of the superconducting quadrupole. Previously, in the section *The Module Data Files: ScQ30.DAT SC Quadrupole Design Data* of the 3MV example, we have given a description of the module data file and we have shown pictures of the superconducting quadrupole. The design parameters for station 70 that HILDA calculates are shown below. They will be more meaningful when reference is made to the pictures in Figure 2-13.

The parameter output is divided into two parts. The first part refers to one individual SC quadrupole. For this quadrupole we have radial dimensions, cross-sectional areas, volumes, weights, and costs of the components. The individual quadrupoles are then bundled together to form an array of quadrupoles. In this case the number of beams is 16; so there are 16 quadrupoles, or channels, in the array. The second part pertains to the quadrupole array; we again show radial dimensions, areas, volumes, weights, and costs.

Following this specification of design parameters is a section that shows the materials used for this design.

The Figure 2-19, *HILDA Output E* is a copy of the output file created by the module StrucCore. It is similar to the previous SC quadrupole file in that it has parameters and data that define the minimum-cost design; in this case the minimum-cost acceleration cell.

The primary output from this module is the variable *dolAStruct*, which is the cost of the acceleration cell. This cell does NOT include the focusing/defocusing quadrupole. This amount is simply the cost of the cell as determined by its component material costs. The design information in this file should be referred to the pictures in the data file StrucCore.DAT. These pictures have been included in this report in the section *The Module Data Files: StrucCore.DAT Acceleration Cell Design Data* of the previously described 3MV example. See Figure 2-14.

The file StrucCore.Out shown in Figure 2-19 contains sections of information. The first section is the dollar cost of the components of the cell; based on the cost of the material used, as obtained from the aforementioned data file. Next are the selection parameters used in this design. These, when associated with the contents of the data file StrucCore.DAT, define the component materials. We next give component weights, followed by cell and core parameters. The cell and core parameters should give enough information to draw a picture of the cell that HILDA has designed.

In Figure 2-20, *HILDA Output F* we show an outline of the remainder of the CostV1.070 file. The data files used by the HILDA modules are copied and recorded in this output file. This is not necessarily redundant information. HILDA is very modular and future versions of HILDA may be designed to modify the data file during execution, to help HILDA find a minimum-cost design. The actual data files that were used are those that are recorded in this output file. A description of these data files appears in the section *The Module Data Files*. Since these files are essentially the same as for the 3MV example, the reader should refer to that example for the pictures that illustrate the components.

The final entry into this logfile is a list of module versions. This is usually of no interest to the user. However, it serves a purpose of verifying the integrity of the HILDA modules. A message that a version file does not agree with the module version should serve as a warning to the user that all may not be well in HILDA.

We conclude this discussion of the HILDA output file CostV1.070 by noting that there are many cross-checks that can be performed using this output. These checks should occasionally be done; to insure that data input mistakes have not been made. Performing these checks also builds confidence that HILDA is indeed calculating correct results and designing reasonable elements. This file is a logfile, and as such it is a record of what went into the calculation and what was produced by the calculation. Unless something is wrong, or the design criteria or program module logic has changed, it should be possible to recreate this output by using the input parameter values recorded in CostV1.070.

Figure 2-15. HILDA Output A

```

FILE: CostV1.070
DATE: 91/10/24
TIME: 15:51:22
I/O variable types:
implicit real      (a-h,o-z)
implicit integer (i-m)

The optimum value was found at:
RL      = 4.0000000E+00 : ! half period      [m]
a       = 2.9999999E-02 : ! beam radius      [m]
delV    = 2.8000000E+06 : ! voltage gain     [V]
eta     = 1.0000000E-01 : ! packing factor   [ ]

The cost is:
costDol  = 6.4759331E+05 : ! cost             [$]
perVoltDol = 2.3128332E-01 : ! voltage gain cost [$ / delV]

The input variables for this solution are:
IDStation = 70 : ! station name
Qsys      = 1.3333330E-03 : ! system charge   [C]
numBeam   = 16 : ! number of beams          [ ]
Amu       = 2.0000000E+02 : ! atomic mass     [amu]
q         = 3.0000000E+00 : ! charge state    [e]
V         = 3.0000000E+03 : ! cumulative voltage [MV]
sig0      = 7.2000000E+01 : ! undepressed tune [deg]
epsn      = 1.0000000E-06 : ! nor. emit., no PI [m-r]

```

Figure 2-16. HILDA Output B

```

Associated quantities are given below:
Process KenVar:
tauI      = 8.3333311E-05 : ! charge per beam   [C]
betaGamma = 3.1456658E-01 : ! beta * gamma     [ ]
gamma     = 1.0483092E+00 : ! Energy/rest Energy [ ]
beta      = 3.0007041E-01 : ! v/c             [ ]
Brho      = 6.5157227E+01 : ! magnetic rigidity [T-m]

Process CostGrPt scanned over the grid:
RLmin     = 4.0000000E+00 : ! min. half period   [m]
RLmax     = 4.0000000E+00 : ! max. half period   [m]
delRL     = 4.0000000E+00 : ! grid interval      [m]
aMin      = 2.9999999E-02 : ! min. beam size     [m]
aMax      = 2.9999999E-02 : ! max beam size      [m]
dela      = 2.9999999E-02 : ! grid interval      [m]
delVmin   = 2.8000000E+06 : ! min. voltage change [V]
delVmax   = 2.8000000E+06 : ! max. voltage change [V]
deldelV   = 2.8000000E+06 : ! grid interval      [V]
etaMin    = 1.0000000E-01 : ! min. packing factor [ ]
etaMax    = 1.0000000E-01 : ! max. packing factor [ ]
delEta    = 1.0000000E-01 : ! grid interval      [ ]

The time required for the grid scan:
numGrPts  = 1.0000000E+00 : ! # of grid points   [ ]
delTime   = 7.0312500E-02 : ! grid scan time     [s]
aveTime   = 7.0312500E-02 : ! ave. case time     [s]

```

Figure 2-17. HILDA Output C

```

Associated intermediate Results:

FILE: CostGrPt.OUT                                VERSION: 910910
DATE: 91/10/24 // 15:51:21
I/O variable types
implicit real (a-h,o-z)
implicit integer (i-m)

IN
  IDStation = 70 ! station for this calculation
  numBeam = 16 ! number of beams
  tauI = 8.333331E-05 ! charge [C]
  sig0 = 7.200000E+01 ! undepressed tune [deg]
  betaGamma = 3.1456658E-01 ! beta * gamma
  beta = 3.0007041E-01 ! v/c [ ]
  Brho = 6.5157227E+01 ! magnetic rigidity [T-m]
  RL = 4.000000E+00 ! half period [m]
  a = 2.999999E-02 ! beam radius [m]
  delV = 2.800000E+06 ! voltage gain [V]
  eta = 1.000000E-01 ! packing factor [ ]

OUT
  perVoltDol = 2.3128332E-01 ! voltage gain cost [$ / delV]

AlSighAbar
  alpha = 5.8725291E-01 ! [ ]
  sigmaH = 3.0344951E+00 ! depressed tune [deg]
  abar = 2.2546498E-02 ! an ave. beam size [m]

QIT
  perv = 1.0956922E-05 ! perv
  current = 3.5322086E+02 ! beam current [A]
  taup = 2.5951650E-07 ! pulse width [s]
  E = 7.000000E+05 ! ave. volt gain/HLP [V/m]
  voltSec = 7.2664618E-01 ! volt seconds [V-s]

ENDFILE: CostGrPt.OUT

```

Figure 2-18. HILDA Output D

```

FILE: ScQ30.OUT                                VERSION: 910910
DATE: 91/10/24 // 15:51:20

I/O variable types
implicit real (a-h,o-z)
implicit integer (i-m)

IN
  a = 3.000000E-02 ! beam edge radius [m]
  RL = 4.000000E+00 ! half period [m]
  eta = 1.000000E-01 ! packing factor [ ]
  Brho = 6.515723E+01 ! magnetic rigidity [T-m]
  sig0 = 7.200000E+01 ! undepressed tune [deg]
  numBeam = 16 ! number of beams [ ]

OUT
  dolTArray = 8.769912E+04 ! cost of quad array [$]
  roTArray = 4.972594E-01 ! outer radius of quad array [m]

```

Figure 2-18. HILDA Output D (continued)

```

Data and calculated Quadrupole parameters:
sig0      = 7.200000E+01  :!undepressed tune           (deg)
rWire     = 6.340000E-02  :!inner wire radius         (m)
rWo       = 7.340000E-02  :!outer wire radius          (m)
drWire    = 1.000000E-02  :!wire thickness           (m)
BWire     = 3.141682E+00  :!field, inner wire radius   (T)
BWo       = 3.637215E+00  :!field, outer wire radius   (T)
Bprime    = 4.955334E+01  :!quad., field gradient      (T/m)
avJ       = 1.317847E+09  :!average current density [A/m**2]
R         = 4.750000E-02  :!aperture radius           (m)
drWrap    = 3.236777E-03  :!quad wrap, thickness       (m)
pitch     = 1.562736E-01  :!beam center to center      (m)
endMag    = 7.340000E-02  :!length of magnet ends      (m)
zlMag     = 5.468000E-01  :!length of magnet iron      (m)
wtScQuad  = 5.329318E+01  :!weight of 1 SC quadrupole   (kg)
costQuad  = 4.757089E+03  :!cost of 1 SC quadrupole    ($)
wtTArray  = 1.316119E+03  :!weight of SC quad Array     (kg)
doltArray = 8.769912E+04  :!cost of SC quad Array      ($)

```

SINGLE SC QUADRUPOLE ASSEMBLY

Radial dimensions

```

drPIC     = 2.590000E-02  :!Pipes, Insulation and Cooling (m)

```

Individual radii

```

rPipe (1) = 4.750000E-02  :!Pipes                      (m)
rPipe (2) = 6.140000E-02  :!Pipes                      (m)
rInsul(1) = 4.950000E-02  :!Insulation                (m)
rInsul(2) = 5.380000E-02  :!Insulation                (m)
rInsul(3) = 5.810000E-02  :!Insulation                (m)
rCool (1) = 5.280000E-02  :!Cooling                    (m)
rCool (2) = 5.710000E-02  :!Cooling                    (m)

```

Areas, xy crosssection

```

xyScQ     = 2.442143E-02  :!One SC quadrupole           (m**2)
xyWrap    = 7.495908E-03  :!Wrap (stress), incl. shell (m**2)
xyCable   = 4.297699E-03  :!Cable, SC + non-SC!       (m**2)
xyPIC     = 5.539599E-03  :!Tot. Pipes, Insul. & Cool. (m**2)
xyVac     = 7.088219E-03  :!Vacuum, Beam pipe         (m**2)

```

Volumes for One SC Quadrupole

```

volScQ    = 1.335364E-02  :!Complete Quad.           (m**3)
volWrap   = 4.098763E-03  :!Wrap (stress), incl. shell (m**3)
volCable  = 2.349982E-03  :!Cable, SC + non-SC       (m**3)
volPipe   = 7.620263E-04  :!Pipes, total             (m**3)
volInsul  = 1.886015E-03  :!Insulation, total        (m**3)
volCool   = 3.810132E-04  :!Cooling, total           (m**3)
volPIC    = 3.029053E-03  :!Tot. Pipes, Insul. & Cool. (m**3)
volVac    = 3.875838E-03  :!Vacuum, Beam Pipe        (m**3)

```

Weights of Components in One SC Quadrupole

```

wtScCab   = 6.377758E+00  :!SC Cable material         (kg)
wtCabS    = 1.344615E+01  :!non-SC Cable material     (kg)
wtCable   = 1.982391E+01  :!Total Cable Weight        (kg)
wtWrap    = 3.346927E+01  :!Outer Wrap, includes shell (kg)
wtPipe    = 0.000000E+00  :!Pipe layers               (kg)
wtInsul   = 0.000000E+00  :!Insulation layers         (kg)
wtCool    = 0.000000E+00  :!Cooling layers            (kg)
wtScQuad  = 5.329318E+01  :!Total of one SC Quadrupole (kg)

```

Figure 2-18. HILDA Output D (continued)

```

Costs of Components in one SC Quadrupole
dScCab  = 1.913327E+03  :!SC Cable material          [$]
dCabS   = 6.723076E+02  :!non-SC Cable material      [$]
dCable  = 2.585635E+03  :!Total Cable, SC + non-SC    [$]
dWrap   = 8.367318E+02  :!Wrap (stress), includes shell [$]

The rest of the SC Quadrupole (channel)
dPipe   = 0.000000E+00  :!Pipes                      [$]
dInsul  = 0.000000E+00  :!Insulation                  [$]
dCool   = 0.000000E+00  :!Cooling                      . [$]
dBalance = 1.334723E+03  :!Pipes, Insulation and Cooling [$]
costQuad = 4.757089E+03  :!Total, one SC Quad. (channel) [$]

COMPLETE ARRAY of numBeam SC Quadrupole channels + Wrap
Radial dimensions
xAScQ   = 6.250942E-01  :!Width of the SC Quads      [m]
yAScQ   = 6.250942E-01  :!Height of the SC Quads     [m]
dxAOtC  = 3.906839E-02  :!Array outer collar,dr      [m]
dyAOtC  = 3.906839E-02  :!Array outer collar,dr      [m]

Areas, xy Cross Section
xyAScQ  = 3.907428E-01  :!SC Quadrupole bundle       [m**2]
xyAVac  = 1.134115E-01  :!Vacuum, Beam Pipes         [m**2]
xyAOtC  = 1.037911E-01  :!Array outer Collar         [m**2]
xyAQuad = 4.945339E-01  :!Array, Total xy Area       [m**2]

Volumes
volAScQ = 2.136582E-01  :!SC Quadrupole bundle       [m**3]
volAVac = 6.201341E-02  :!Vacuum, Beam Pipes         [m**3]
volAOtC = 5.675295E-02  :!Array outer Collar         [m**3]
volAQuad = 2.704111E-01 :!Array Total xyz space     [m**3]

Weights of Array Components (Total for numBeam SC Quads.)
wtAScCab = 1.020441E+02 :!SC Cable material          [kg]
wtACabS  = 2.151384E+02 :!non-SC Cable material      [kg]
wtACable = 3.171825E+02 :!Total Cable Weight        [kg]
wtAWrap  = 5.355083E+02 :!Outer Quad. Wraps (stress)    [kg]
wtAPipe  = 0.000000E+00 :!Pipes                      [kg]
wtAInsul = 0.000000E+00 :!Insulation layers          [kg]
wtACool  = 0.000000E+00 :!Cooling layers            [kg]
wtAScQ   = 8.526909E+02 :!SC Quadrupoles            [kg]
wtAOtC   = 4.634276E+02 :!Outer Array collar         [kg]
wtTArray = 1.316119E+03 :!Total of the Complete Array [kg]

Costs of Array Components (Total for numBeam SC Quads.)
dAScCab  = 3.061324E+04 :!SC Cable material          [$]
dACabS   = 1.075692E+04 :!non-SC Cable material      [$]
dACable  = 4.137016E+04 :!Total for cable, SC + non-SC [$]
dAWrap   = 1.338771E+04 :!Quad Wraps (stress)        [$]
dAPipe   = 0.000000E+00 :!Pipes                      [$]
dAInsul  = 0.000000E+00 :!Insulation layers          [$]
dACool   = 0.000000E+00 :!Cooling layers            [$]
dABalance = 2.135557E+04 :!Total for Pipes, Insul., Cool. [$]
dAQuad   = 7.611343E+04 :!Total for the SC Quads      [$]
dAOtC    = 1.158569E+04 :!Outer Array collar         [$]
doltArray = 8.769912E+04 :!Total for Array Assembly   [$]

```

Figure 2-18. HILDA Output D (continued)

```

DATA LOADED FROM FILE ScQ30.DAT

CHOOSE MATERIALS for the SC Quadrupole
iScCab = 1           !! SC wires [ ]
iCabS = 1           !! non-SC wire space [ ]
iWrap = 1           !! Wrap (stress) [ ]
iPipe = 1           !! vac. & outer Quad pipes [ ]
nPipe = 2           !! # of pipes [ ]
drPipe = 2.000000E-03 !! thickness of pipes [m]
iInsul = 1          !! insulation layers [ ]
nInsul = 3          !! # of layers [ ]
drInsul = 3.300000E-03 !! thickness of layers [m]
iCool = 1           !! cooling layers [ ]
nCool = 2           !! # of layers [ ]
drCool = 1.000000E-03 !! thickness of layers [m]

CHOOSE MATERIAL for the Array of numBeam Quadrupoles
iAWrap = 1           !! outer wrap (collar) [ ]

SET LIMITS and PARAMETER VALUES

Quadrupoles
drWmin = 1.000000E-02 !! minimum wire thickness [m]
drWmax = 1.000000E-01 !! maximum wire thickness [m]
dShell = 1.500000E-03 !! outer shell thickness [m]

Non-Quadrupole free space
zQend = 0.000000E+00 !! end packaging for the quads. [m]
fZSpace = 1.000000E-01 !! non-magnet space limit, frac.of L [ ]

Coefficients for the Superconducting Quadrupole Calculation:
cR1 = 1.250000E+00 !! aperture radius = R = cR1*a+cR2 [ ]
cR2 = 1.000000E-02 !! [m]

Quadrupole Wrap (stress) scaling parameters
sWrap = 1.000000E-02 !! wrap thickness for BWrap, rWrap [m]
BWrap = 5.000000E+00 !! field for scaling the Quad. Wrap [T]
rWrap = 1.200000E-01 !! radius used with BWrap [m]

Quadrupole SC wire inner radius limit
rWtRL = 5.000000E-01 !! rWire .lt. eta*RL [m]

Array of numBeam Quadrupoles
fCollar = 2.500000E-01 !! wrap (collar) width, frac. of pitch [ ]

Quadrupole (channel) Assembly-Complexity Cost factor
BFactor = 3.900000E-01 !! times the Quad (Wrap + Cable) cost [ ]

Array Assembly-Complexity Cost factor
cAFact = 1.000000E+00 !! times the Array cost [ ]

MATERIAL USED in the SC Quadrupole Array

Superconducting Cable
iScCab = 1           !! ID # of data set
IDScMat = 'NbTi Niobium Titanium' !!
denScMat = 7.600000E+03 !! material density [kg/m**3]
uCSMat = 3.000000E+02 !! unit cost [$ /kg]
qFScMat = 1.000000E+00 !! quantity factor [ ]
cFScMat = 1.000000E+00 !! complexity factor [ ]

Current density parameters and Field limits
cJCoeff = 2.900000E+09 !! slope of cJ curve [A/m**2]
cJBn = 1.000000E+01 !! numerator B parameter [T]
cJBd = 5.000000E+00 !! denominator B parameter [T]
rLamda = 3.571000E-01 !! S C wire pack. fraction [T]
BWomax = 1.000000E+01 !! B at outer wire, maximum [T]

```


Figure 2-18. HILDA Output D (continued)

```

Non SC Cable
iCabS      = 1           !! ID # of data set
IDCMat     = 'Cu   Copper'      '  :!
denCMat    = 8.900000E+03  !! material density      [kg/m**3]
uCCMat     = 5.000000E+01  !! unit cost              [$ /kg]
qFCMat     = 1.000000E+00  !! quantity factor       [ ]
cFCMat     = 1.000000E+00  !! complexity factor      [ ]

Quadrupole Wrap (Stress)
iWrap      = 1           !! ID # of data set
IDWMat     = 'Steel'        '  :!
denWMat    = 8.165700E+03  !! material density      [kg/m**3]
uCWMat     = 2.500000E+01  !! unit cost              [$ /kg]
qFWMat     = 1.000000E+00  !! quantity factor       [ ]
cFWMat     = 1.000000E+00  !! complexity factor      [ ]

Array Outer Wrap (Collar)
iAWrap     = 1           !! ID # of data set
IDAotMat   = 'Steel'        '  :!
denAotMat  = 8.165700E+03  !! material density      [kg/m**3]
uCAotMat   = 2.500000E+01  !! unit cost              [$ /kg]
qFAotMat   = 1.000000E+00  !! quantity factor       [ ]
cFAotMat   = 1.000000E+00  !! complexity factor      [ ]

Pipes , Vacuum and around Quad outer insulation layer
iPipe      = 1           !! ID # of data set
IDPMat     = 'No Data on this Material'      '  :!
denPMat    = 0.000000E+00  !! material density      [kg/m**3]
uCPMat     = 1.000000E+00  !! unit cost              [$ /kg]
qFPMat     = 1.000000E+00  !! quantity factor       [ ]
cFPMat     = 1.000000E+00  !! complexity factor      [ ]

Insulation (thermal)
iInsul     = 1           !! ID # of data set
IDIMat     = 'No Data on this Material'      '  :!
denIMat    = 0.000000E+00  !! material density      [kg/m**3]
uCIMat     = 1.000000E+00  !! unit cost              [$ /kg]
qFIMat     = 1.000000E+00  !! quantity factor       [ ]
cFIMat     = 1.000000E+00  !! complexity factor      [ ]

Cooling Layers, data sets
iCool      = 1           !! ID # of data set
IDCLay     = 'No Data on this Material'      '  :!
denCLay    = 0.000000E+00  !! material density      [kg/m**3]
uCCLay     = 1.000000E+00  !! unit cost              [$ /kg]
qFCLay     = 1.000000E+00  !! quantity factor       [ ]
cFCLay     = 1.000000E+00  !! complexity factor      [ ]

ENDFILE: ScQ30.OUT

```

Figure 2-19. HILDA Output E

```

FILE: StrucCore.OUT                                VERSION: 910910
DATE: 91/10/24 // 15:51:21

I/O variable types
implicit real    (a-h,o-z)
implicit integer (i-m)

IN
  voltSec   = 7.2664618E-01    :!volt-sec per half period    [V-s]
  numBeams  = 16               :!number of beams           [ ]
  roTArray  = 4.9725944E-01    :!outer radius, quad array [m]
  voltGain  = 2.8000000E+00    :!peak acc. per half period [MV]
  halfPeriod = 4.0000000E+00   :!lattice half-period length [m]

OUT
  dolAStruct = 5.598942D+05    :!HLP cost of acc. structure [$]

The intermediate values calculated and returned are:
COST OF COMPONENTS
  numdolHlp = 6               :!number of costs returned
  dolAm      = 4.668889D+04    :!core amorphous material    [$]
  dolCorH    = 6.373007D+04    :!core housing               [$]
  dolCelH    = 4.489275D+04    :!cell housing               [$]
  dolCCI     = 8.903680D+03    :!core/cell housing insulation [$]
  dolGI      = 3.934876D+05    :!gap insulator              [$]
  dolDiC     = 2.191167D+03    :!dielectric coolant         [$]

MATERIALS USED: flags select material from StrucCore.dat
  numidMat   = 8              :!# of used material flags returned
  idAm       = 1              :!core winding amorphous tape
  idWTape    = 4              :!core tape width
  idCSM      = 2              :!core submodule housing
  idCH       = 2              :!cell housing
  idIN       = 2              :!core/cell insulation
  idDiC      = 4              :!dielectric coolant
  idGP       = 6              :!gap vacuum pressure range
  idGI       = 1              :!gap insulator

COMPONENT WEIGHTS:
  numwthlp   = 20             :!number of weights that are returned

Half Lattice Period weights for the acceleration structure
  wtAStruct  = 2.298129D+04    :!HLP weight of acc. structure [kg]
  wtAm       = 9.337778D+03    :!core amorphous material    [kg]
  wtCorH     = 6.438227D+03    :!core housing               [kg]
  wtCelH     = 4.535217D+03    :!cell housing               [kg]
  wtCCI      = 4.313443D+02    :!core/cell housing insulation [kg]
  wtGI       = 9.187461D+02    :!gap insulator              [kg]
  wtDiC      = 1.319980D+03    :!dielectric coolant         [kg]

```

Figure 2-19. HILDA Output E (continued)

Sub component weights				
wtCore	=	6.669841D+02	!!amor. material, per core	[kg]
wtCEP	=	5.073117D+01	!!housing end plate, per core	[kg]
wtCOH	=	1.989284D+02	!!outer housing, per core	[kg]
wtCOB	=	1.594825D+02	!!inner bobbin, per core	[kg]
wtCHPC	=	4.598733D+02	!!housing total, per core	[kg]
wtCEI	=	5.714488D+00	!!end plate ins. (2 per core)	[kg]
wtCOI	=	1.096856D+01	!!outer housing (ins./core)	[kg]
wtCI	=	2.239754D+01	!!insulation total, per core	[kg]
wtCHIE	=	8.412764D+00	!!cell hous. (end ins. 1/core)	[kg]
wtCHI	=	8.412764D+00	!!cell hous. insulation, total	[kg]
wtHSGEP	=	3.985775D+02	!!cell hous. end plate, pair	[kg]
wtOHR	=	3.440581D+03	!!cell hous. supp. ring	[kg]
wtHSGGI	=	6.960581D+02	!!gap insulator support ring	[kg]
CELL/CORE PARAMETERS:				
numCP	=	29	!!# of parameters returned	
numCore	=	1.400000D+01	!!# of cores & PFNs per cell	[]
cellL	=	3.591196D+00	!!z axis, cell length	[m]
gIL	=	2.333333D+00	!!gap insulator length	[m]
pV	=	2.000000D-01	!!required PFN peak per core	[MV]
tI	=	3.000000D-02	!!gap insulator width	[m]
QA1	=	4.088040D-01	!!support lgth., quadrupole, etc.	[m]
zIH	=	2.000000D-02	!!cell housing widths, z axis	[m]
zIPH	=	1.000000D-02	!!core housing widths, z axis	[m]
zICi	=	1.015228D-02	!!z length, core insulator	[m]
delTC	=	2.000000D-02	!!core housing top	[m]
delBC	=	2.000000D-02	!!core housing bottom	[m]
delCH	=	2.309401D-02	!!cell housing top	[m]
riH	=	5.469854D-01	!!inside radius to acc. gap insul.	[m]
delRg	=	5.128205D-02	!!radial cell-acc. gap	[m]
riM	=	6.282674D-01	!!core housing inner radius	[m]
riC	=	6.482674D-01	!!core inside radius	[m]
roC	=	7.759823D-01	!!outside radius of core	[m]
roCore	=	8.061345D-01	!!core housing outer radius	[m]
roHSG	=	8.292286D-01	!!cell housing outer radius	[m]
acAM	=	2.906585D-01	!!acc. cell amor. mat. area	[m**2]
pFam	=	8.000000D-01	!!packing fraction	[]
aC	=	2.595165D-02	!!core cross sectional area	[m**2]
wTape	=	2.032000D-01	!!core amor. mat.tape width	[m]
hC	=	1.277148D-01	!!height of amorphous material	[m]
aRC	=	6.285177D-01	!!core amor. mat. h/w ratio	[]
fQ	=	1.100000D+00	!!quad array per. supp. factor	[]
gILmax	=	2.872957D+00	!!max. gap ins. length	[m]
aRCmax	=	4.000000D+00	!!max. hth/width ratio, core	[]
QA1min	=	4.000000D-01	!!min. quad. support length	[m]
MATERIAL DATA Values actually used:				
Core amorphous material				
denAm	=	7.180000E+00	!!density	[gm/cm**3]
cFacAm	=	1.000000E+00	!!complexity factor	[]
qFacAm	=	1.000000E+00	!!quantity factor	[]
uCostAm	=	5.000000E+00	!!unit cost	[\$/kg]
pFam	=	8.000000E-01	!!radial packing fact.....	[]
delBAm	=	2.500000E+00	!!flux swing	[T]
Core tape widths				
wTape	=	2.032000E+01	!!core tape width	[cm]
aRCmax	=	4.000000E+00	!!max. height/width	[]

Figure 2-19. HILDA Output E (continued)

```

Core Sub Module housing material
denCSM = 8.165700E+00      :!density          [gm/cm**3]
cFacCSM = 1.000000E+00      :!complexity factor      []
qFacCSM = 1.000000E+00      :!quantity factor        []
uCostCSM = 9.898700E+00      :!unit cost              [$ /kg]

Cell Housing material
denCH = 8.165700E+00      :!density          [gm/cm**3]
cFacCH = 1.000000E+00      :!complexity factor      []
qFacCH = 1.000000E+00      :!quantity factor        []
uCostCH = 9.898700E+00      :!unit cost              [$ /kg]
emCH = 3.000000E+01      :!elasticity modulus [10**6 lb/in**2]

Core & cell Housing Insulation
denIN = 9.850000E-01      :!density          [gm/cm**3]
cFacIN = 1.000000E+00      :!complexity factor      []
qFacIN = 1.000000E+00      :!quantity factor        []
uCostIN = 2.064170E+01      :!unit cost              [$ /kg]
bVoltIN = 1.970000E+02      :!break down voltage    [kV/cm]

Gap Insulator
denGI = 3.717000E+00      :!density          [gm/cm**3]
cFacGI = 1.000000E+00      :!complexity factor      []
qFacGI = 1.000000E+00      :!quantity factor        []
uCostGI = 4.282876E+02      :!unit cost              [$ /kg]
bVoltGI = 1.200000E+01      :!break down voltage    [kV/cm]

Dielectric coolant
denDiC = 1.800000E+00      :!density          [gm/cm**3]
cFacDiC = 1.000000E+00      :!complexity factor      []
qFacDiC = 1.000000E+00      :!quantity factor        []
uCostDiC = 1.660000E+00      :!unit cost              [$ /kg]
bVoltDiC = 3.900000E+01      :!break down voltage    [kV/cm]

Acc. Gap Voltage break down strength
bVRGap = 0.500000E+02      :!                    [kV/cm]

ENDFILE:  StrucCore.OUT

```

Figure 2-20. HILDA Output F

```

DATA FILES USED:
See the 3000MV example in the HPD disks

PROCESS VERSIONS:
See the 3000MV example in the HPD disks

ENDFILE:  CostV1.010

```

The Module Data Files

Each HILDA data file discussed below is associated with a HILDA module of the same name. The module reads the file *module name.DAT* to get the data it requires. For example module ScQ30 reads parameter data from the file ScQ30.DAT.

CalCost1.DAT Beam Parameter File

It is necessary to define the beam parameters for each station that the user intends to use HILDA to find a minimum-cost design. Those parameter values are set in this data file which is subsequently read by the HILDA module CalCost. In Figure 2-21, *CalCost1.DAT* below we show the contents of that file as used in this example.

This particular data file can contain data for more than one station; for the 4MJ driver example it contained data for seven stations. The example we are presenting here produces a design for the seventh station, which we have identified as station 70. This data set could have been used for the 3MV example, since it contains beam data for the 3MV station. Also this data set could also be shortened to contain only the beam data for station 70, the 3000MV station.

We should note that HILDA is very modular in design. Future versions of HILDA can in principal have modules that modify this data file to find what beam parameter values give the minimum-cost design of a driver. For example, the number of beams could be adjusted. We could easily do that right now with this version by noting that the station number is simply a way of identifying a data packet. If we wished we could add more data packets for a given point along the machine and then compare the cost of the various designs that HILDA produced.

The total amount of charge in the beam at this station is 0.00133333 Coulombs. This is transported in 16 beamlets. The particles that make up the beam have 200 atomic mass units and are ions with a positive charge state of 3. At this station the cumulative acceleration voltage that the machine has supplied is 3000 million volts. The undepressed tune (single particle, no space charge) of the beam is 72 degrees. The normalized emittance of each beam in the 16 beams is 0.000001π [meter-radians].

In the file CalCost1.DAT we also have a file name, which in this example is shown as *CostV1.070*. The name furnished before the run is immaterial. In some data files you may see the file name set to null. HILDA enters the name of the file that contains the output for the minimum-cost design at the station. In this case the name became CostV1.070. If no minimum-cost design is found, this file name is not necessarily updated; the program asks the user what to do.

CostV1.DAT Parameter Search Space

This data file is associated with the module CostV1 that does the parameter space scan. It is different from the 3MV example data set only in the values of the parameters. The space that is scanned over is limited, here, to one point; the minimum-cost design parameter values. The structure half-period *RL*, the maximum beam radius *a*, the voltage gain *delV* that the station supplies, and the quadrupole packing fraction *eta* are set by the data in this file. At each point (*RL,a,delV,eta*) in this space HILDA determines if the beam can be transported. If it cannot be transported, then HILDA will NOT try to find a minimum-cost design; it will skip to the next point in the parameter space. The data file CostV1.DAT, shown in Figure 2-22, *CostV1.DAT*, defines the parameter space by setting the limits and the number of points for each parameter. In this particular example we limit the search to one point.

As we mentioned before, the contents of the logfile CostV1.LOG can vary. This is controlled by setting the parameter *iLog* in this data file.

AlSighBar.DAT Beam Dynamics Parameters

HILDA solves equations that pertain to the dynamical beam transport problem. Information about the model is in the section *The HILDA Model: Beam Transport Equations* of this report. We note here that the parameters in this data are the same for all the stations in the 4MJ driver example from which this example was extracted. See Figure 2-23, *AlSighAbar.DAT*.

TranMod.DAT Transport Module Selection Data

HILDA first asks whether the beam can be transported. For those points in the parameter space for which the beam CAN be transported HILDA designs appropriate elements. The module TranMod reads data that selects which design modules to use. Both stations 10 and 70 use SC quadrupoles, so this data set is the same for both stations; all the 4MJ driver stations use SC quadrupoles. In principal, this data could have been set to choose an Fe quadrupole. See Figure 2-24, *TranMod.DAT*.

FeQ20.DAT Fe Quadrupole Design Data

The HILDA module FeQ20 designs an iron quadrupole. This data set furnishes all the necessary design information. The module FeQ20 is never used for this example, so this data set is not required to be present. See Figure 2-25, *FeQ20.DAT*.

ScQ30.DAT SC Quadrupole Design Data

The module ScQ30 designs a superconducting quadrupole. This module is similar to the module FeQ20. The ScQ30.DAT data set furnishes information needed for the superconducting quadrupole. This data set is identical to the one used in the 3MV station. We thus refer the reader to the description in that example. See Figure 2-26, *ScQ30.DAT*.

We note, however, that the CostV1.070 file is the final arbitrator as to what parameter values are used for the SC quadrupole file. It happens that for the examples given here we do not change the selection of materials; however, it is possible to do that. The information written in the CostV1.070 logfile will contain the values used for the design that is recorded in that file.

StrucCore.DAT Acceleration Cell Design Data

The module StrucCore designs an acceleration-transport module. The basic data for this module is in the file StrucCore.DAT. This data set is identical to the one used in the 3MV station. See Figure 2-27, *StrucCore.DAT*. We thus refer the reader to description in that example. We should note, however, that both the 3MV and the 3000MV examples have been extracted from the 4MJ driver example. Finding the minimum-cost design requires much more effort than is implied here. For example, both the 3MV and the 3000MV examples use the same amorphous tape width. However, to determine which of the tape widths yields a minimum-cost design requires that we run HILDA with the available tape widths. The present version of HILDA requires that the StrucCore.DAT file be correctly set for each tape width that HILDA investigates; the user must do this. Future versions of HILDA will contain modules that easily set such parameters. It is rather straight forward to build modules that automatically search over the tape widths and choose for us the correct tape width. The modular construction of HILDA can accommodate many useful extensions.

Figure 2-21. CalCost1.DAT

```
c  FILE: CalCost1.DAT
c  DATE: 910922
```

```
c  EXAMPLE: dr4MJ @ 3000MV
```

At each station this data defines the basic parameters of the beam that is being transported. Upon completion of the cost minimization calculation at the station, the file CostV1.nnn, where nnn is the value of IDStation, will contain all the design information.

Basic Beam Data for 4MJ driver at 3.0 MV

varName	varValue	data type	comment
IDStation =	10	:integer	! station identification #
Qsys =	1.333333e-3	:real	! system charge [C]
N =	16	:integer	! number of beams []
Amu =	200	:real	! atomic no. [amu]
q =	3	:real	! charge state []
V =	3	:real	! cumulative voltage [MV]
sig0 =	72.0	:real	! undepressed tune [deg]
epsn =	1e-6	:real	! nor. emittance [m-r]
filename =	'null'	:character	! name of file

Basic Beam Data for 4MJ driver at 10.0 MV

varName	varValue	data type	comment
IDStation =	20	:integer	! station identification #
Qsys =	1.333333e-3	:real	! system charge [C]
N =	16	:integer	! number of beams []
Amu =	200	:real	! atomic no. [amu]
q =	3	:real	! charge state []
V =	10	:real	! cumulative voltage [MV]
sig0 =	72.0	:real	! undepressed tune [deg]
epsn =	1e-6	:real	! nor. emittance [m-r]
filename =	'null'	:character	! name of file

Basic Beam Data for 4MJ driver at 30.0 MV

varName	varValue	data type	comment
IDStation =	30	:integer	! station identification #
Qsys =	1.333333e-3	:real	! system charge [C]
N =	16	:integer	! number of beams []
Amu =	200	:real	! atomic no. [amu]
q =	3	:real	! charge state []
V =	30	:real	! cumulative voltage [MV]
sig0 =	72.0	:real	! undepressed tune [deg]
epsn =	1e-6	:real	! nor. emittance [m-r]
filename =	'null'	:character	! name of file

Figure 2-21. CalCost1.DAT (continued)

Basic Beam Data for 4MJ driver at 100.0 MV			
varName	varValue	data type	comment
IDStation =	40	:integer	! station identification #
Qsys =	1.333333e-3	:real	! system charge [C]
N =	16	:integer	! number of beams []
Amu =	200	:real	! atomic no. [amu]
q =	3	:real	! charge state []
V =	100	:real	! cumulative voltage [MV]
sig0 =	72.0	:real	! undepressed tune [deg]
epsn =	1e-6	:real	! nor. emittance [m-r]
filename =	'null'	:character	! name of file
Basic Beam Data for 4MJ driver at 300.0 MV			
varName	varValue	data type	comment
IDStation =	50	:integer	! station identification #
Qsys =	1.333333e-3	:real	! system charge [C]
N =	16	:integer	! number of beams []
Amu =	200	:real	! atomic no. [amu]
q =	3	:real	! charge state []
V =	300	:real	! cumulative voltage [MV]
sig0 =	72.0	:real	! undepressed tune [deg]
epsn =	1e-6	:real	! nor. emittance [m-r]
filename =	'null'	:character	! name of file
Basic Beam Data for 4MJ driver at 1000.0 MV			
varName	varValue	data type	comment
IDStation =	60	:integer	! station identification #
Qsys =	1.333333e-3	:real	! system charge [C]
N =	16	:integer	! number of beams []
Amu =	200	:real	! atomic no. [amu]
q =	3	:real	! charge state []
V =	1000	:real	! cumulative voltage [MV]
sig0 =	72.0	:real	! undepressed tune [deg]
epsn =	1e-6	:real	! nor. emittance [m-r]
filename =	'null'	:character	! name of file
Basic Beam Data for 4MJ driver at 3000.0 MV			
varName	varValue	data type	comment
IDStation =	70	:integer	! station identification #
Qsys =	1.333333e-3	:real	! system charge [C]
N =	16	:integer	! number of beams []
Amu =	200	:real	! atomic no. [amu]
q =	3	:real	! charge state []
V =	3000	:real	! cumulative voltage [MV]
sig0 =	72.0	:real	! undepressed tune [deg]
epsn =	1e-6	:real	! nor. emittance [m-r]
filename =	'CostV1.070'	:character	! name of file

Figure 2-21. CalCost1.DAT (continued)

There is one (1) group of the above data for each of the stations in the system. The actual order of the data is immaterial; however, the values that follow the parameter *IDStation* are for that *IDStation*. Thus each data group contains 8 values as shown above. The data consists of a variable name terminated by at least one trailing blank, then an = sign followed by the value of the variable, followed by the data type represented as :<type>. The variable names are as shown above. The value of each data item is represented by a valid Fortran constant. In particular a 'text string' is used for the string text that names the file identifier of the file containing the associated parameter values found when calculating the minimum cost.

c **ENDFILE: CalCost1.DAT**

Figure 2-22. CostV1.DAT

c **FILE: CostV1.DAT**

c **DATE: 910923**

c **EXAMPLE: dr4MJ @ 3000MV**

c **Parameters that are assigned values from a data file:**

c At the current station, HILDA cycles through the points

c in the parameter space defined below.

Beam Parameter Range Data for 4MJ driver at 3000.0 MV

* varName	varValue	data type	comment
c IDStation	70	:integer	! station identification #
RLmin	= 4.00	:real	! min. structure half-period [m]
RLmax	= 4.00	:real	! max. structure half-period [m]
numRL	= 1	:real	! # of grid points []
aMin	= 0.03	:real	! min. beam size (max) [m]
aMax	= 0.03	:real	! max. beam size (max) [m]
numa	= 1	:real	! # of grid points []
delVmin	= 2800.OE3	:real	! min. voltage gain [V]
delVmax	= 2800.OE3	:real	! max. voltage gain [V]
numDelV	= 1	:real	! # of grid points []
etaMin	= 0.10	:real	! min. quad. packing fraction []
etaMax	= 0.10	:real	! min. quad. packing fraction []
numEta	= 1	:real	! # of grid points []

c **Log File printing is set using the value of iLog.**

iLog	= 3	! 0 OK designs logged to the terminal, skipped points not logged
		! 1 OK and skipped designs logged to the terminal
		! 2 OK and skipped designs logged to CostV1.LOG
		! 3 OK designs logged to CostV1.LOG
		! 4 ONLY Minimum designs logged to CostV1.LOG
		! 5 NO designs logged to CostV1.LOG or to the terminal

c **ENDFILE: CostV1.DAT**

Figure 2-23. AlSighAbar.DAT

```
c FILE: AlSighAbar.DAT
c DATE: 900420
c EXAMPLE: dr4MJ@3000MV
  This is the same as for the 3MV example. See Figure 2-10.
c ENDFILE: AlSighAbar.DAT
```

Figure 2-24. TranMod.DAT

```
c FILE: TranMod.DAT
c DATE: 910923
c EXAMPLE: dr4MJ@3000MV
  This is the same as for the 3MV example. See Figure 2-11.
c ENDFILE: TranMod.DAT
```

Figure 2-25. FeQ20.DAT

```
c FILE: FeQ20.DAT
c DATE: 910923
c EXAMPLE: dr4MJ@3000MV
  This is the same as for the 3MV example. However this 3000MV example does
  not use an Fe quadrupole, so the module FeQ20 is never called. This means
  that this data set is never read. See Figure 2-12.
c ENDFILE: FeQ20.DAT
```

Figure 2-26. ScQ30.DAT

```
c FILE: ScQ30.DAT
c DATE: 910923
c EXAMPLE: dr4MJ@3000MV
  This is the same as for the 3MV example. See Figure 2-13.
c ENDFILE: ScQ30.DAT
```

Figure 2-27. StrucCore.DAT

```
c FILE: StrucCore.DAT
c DATE: 910923
c EXAMPLE: dr4MJ@3000MV
  This is the same as for the 3MV example. See Figure 2-14.
c END: StrucCore.DAT
```

C. HILDA MODULE DATA FILES

The HILDA modules can, when needed, read one or more associated data files. These files are fully formatted and as such are meant to be self-explanatory. In this section we give a short explanation of these data files.

HILDA reads the data using HILDA utility modules. It then creates a temporary file that contains only the data that the module needs; all comments have been removed. The module reads the temporary data file and that file is then deleted. Examples of the HILDA module data files are given above in this report. The reader should refer to one of these data files. Also the complete data files for the examples are in the example folders *dr4MJ @ 3MV*, *dr4MJ @ 3000MV*, and *dr4MJ Data* that are on the HPD disks. A guide to these disks is in the *Appendix*.

The actual data line is of the following form:

Variable name = *value* : data type ! comment [units]

A potential data line cannot contain a c, C, or * in column 1. There is an exception: a double asterisk ** in column 1 and 2 can be a potential data line. Any line in a data file that is NOT a potential data line is simply skipped by the HILDA input routines. If a line is a potential data line, the HILDA input routines look for an equal sign, =. If the line does not contain an = sign, it is skipped. If the line contains an equal sign the data value is written to the temporary file for the module to read. The data value is the information that is to the right of the = sign and to the left of the : symbol.

The data value is assumed to be a valid FORTRAN constant that agrees in type with the type specified in the data line. If no data type is specified, the FORTRAN implicit convention is used for numbers. Those variables that begin with *i, j, k, l, m, n* are integers; the rest are real numbers. The data values in the data lines are written into the temporary file in the same order as they appear in the data file. This order is the order that the module reads them and should NOT be changed.

The variable name that appears in the data file is the name of the variable in the module to which the value is assigned. HILDA has input routines that can find data by the name of the variable. When a module uses such a routine, the data value is assigned to the named variable; the data type is determined by the information after the colon sign, :.

From the above discussion it is obvious that for some HILDA modules the only necessary information in the data line is the value of the variable. For others the variable name, value, and data type are required. The data files in the examples have the correct information, as required by the associated modules. We thus strongly recommend that the user modify ONLY the data values in the data lines.

The lines that are not data lines are simply comments that explain the data; they can be deleted, or more comment lines can be added to the data in the file. The temptation is to eliminate all such comment lines. However, HILDA does not spend much time reading data and it is often the case that the ONLY record of what the data means is in the comments in the data file. We also point out that the data file will appear in the HILDA CostV1.NNN logfile and as such serves as a record of the parameter values used in the minimum-cost design. The information in the comments can be quite informative in that context.

One more comment should be made. Future versions of HILDA will produce files that can be read by programs that help in analyzing the output. A well-formatted data file, with comments that explain the contents, is very useful when generating a report about the HILDA run.

D. ANALYZING HILDA RESULTS

HILDA finds a minimum-cost design and generates output that defines the design. This can be done at different stations along the machine, and it can be done at the same station with different parameter values. The primary purpose of HILDA is to find the cost of the design. There are available programs such as spreadsheets that can analyze the results, compare results at different stations, make estimates and produce graphs. They can also produce well formatted reports. Future versions of HILDA should expand its capability to prepare input files for such programs, rather than trying to duplicate their functions.

There is included in the HPD disks, in the folder *HILDA on ILSE.*, the ILSE example. This example illustrates how a spreadsheet can be used with to analyze the results produced by HILDA. The report in that folder used the application EXCEL by Microsoft Corporation.

3. THE HILDA MODEL

A. INTRODUCTION

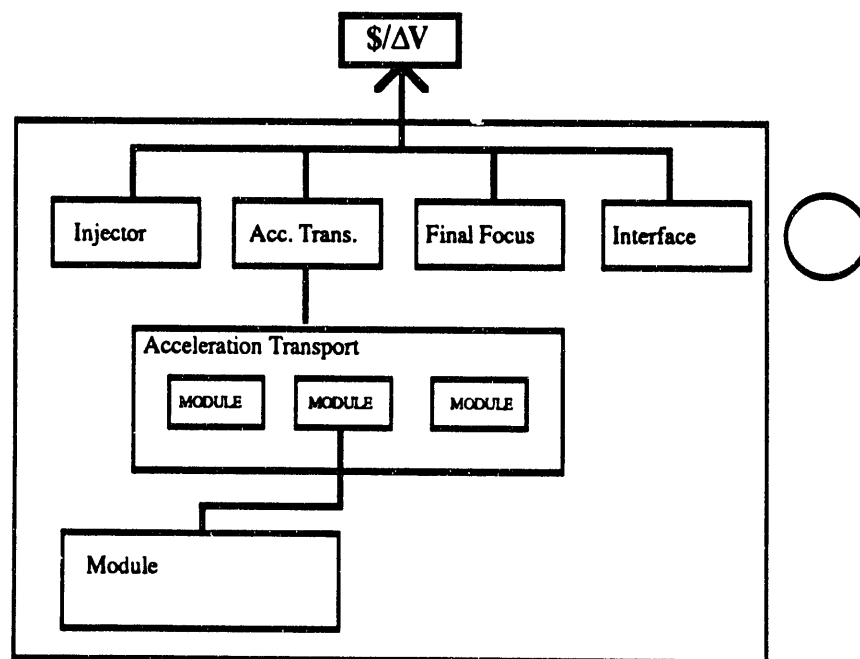
In this section of the Hilda Program Document we describe the model that furnishes the basis of the HILDA cost calculation.

As modeled by HILDA, a Heavy Ion Linear Induction Driver consists of an injector, an acceleration-transport section, a final focus drift-compression section, and an interface to the target chamber. The initial particle beam for the driver is furnished by the injector. After leaving the injector, induction modules with specified gap voltages ΔV accelerate the beam to the design energy. As the beam gains energy and moves through the system, quadrupole focusing elements are used to keep the beam focused. This is done in the acceleration-transport section. The final focus section compresses and focuses the beam; as required by the target in the reactor chamber.

At stations selected along the acceleration-transport section HILDA designs acceleration modules and focusing elements. For each design HILDA estimates the cost of the complete module. The principal quantity that HILDA furnishes is the dollar cost per voltage gain ($\$/\Delta V$) at a selected station. See Figure 3-1, *Acceleration-Transport Module Cost*. The total cost of the acceleration-transport section can be estimated by selecting a suitable number of stations along the machine, calculating the cost of the module at each of these stations and then calculating the total integrated cost. The cost of the driver without the reactor chamber can be obtained by adding the cost of the injector, the final focus section, and the interface to the reactor chamber.

The present version of HILDA does not have modules that estimate costs in sections other than the acceleration-transport section. Because of its modular construction a future version of HILDA could easily accommodate such modules.

Figure 3-1. Acceleration-Transport Module Cost



B. BEAM TRANSPORT EQUATIONS

A station in the acceleration-transport section of the driver is selected by the user. At this station HILDA designs an acceleration-transport module, which consists of an acceleration cell and quadrupole focusing elements, and then returns the cost of that module. HILDA determines whether the particle beam can be transported before designing the module. If the beam cannot be transported there is no reason to proceed with the module design.

The model that HILDA uses to determine whether the beam can be transported is described in this section.

For each station the user furnishes the primary beam parameters as shown below.

<u>Name</u>	<u>value</u>	<u>Description</u>	<u>Units</u>
Qsys	1.33E-3	total system charge	[C]
N	16	the number of beams	
Amu	200	number of atomic mass units	[amu]
q	3	beam ion charge state	
V	3	total voltage gain at the current station	[MV]
sig0	72	undepressed tune, no space charge	[deg]
epsn	1.0E-6	normalized emittance, no π	[m-r]

The parameter values that we describe as user input, i.e., furnished by the user, can be found in the data sets of the examples furnished in this report; the parameter values have been taken from the 3MV example. See Example 1 of this report.

After this data has been supplied it is necessary to specify the limits for the parameters that HILDA will vary as it proceeds to search for a minimum-cost design. The table below identifies the parameters and shows values taken from the 3MV example in this report.

<u>Name</u>	<u>value</u>	<u>Description</u>	<u>Units</u>
RLmin	0.35	min. structure half-period	[m]
RLmax	0.35	max. structure half-period	[m]
numRL	1	number of points	
aMin	0.035	min. beam size (maximum)	[m]
aMax	0.035	max. beam size (maximum)	[m]
numa	1	unumber of points	
delVmin	7000	min. voltage gain	[V]
delVmax	7000	max. voltage gain	[V]
numDelV	1	number of points	

etaMin	0.40	min. quadrupole packing fraction	[]
etaMax	0.40	max. quadrupole packing fraction	[]
numEta	1	number of points	

The search grid parameter values shown here restrict HILDA to exactly one point in the parameter space; this would not be the case when finding a minimum-cost design.

We have used the HILDA names for the parameters in the above tables. In the rest of this section we will use the following:

<u>HILDA</u>	<u>Equations</u>	<u>Description</u>	<u>Units</u>
Qsys	Q	total system charge	[C]
N	N	the number of beams	
Amu	A	number of atomic mass units	[amu]
q	q	beam ion charge state	
V	V	cummulative voltage at the current station	[MV]
sig0	σ_0	undepressed tune, no space charge	[r]
epsn	ϵ_n	normalized enittance, no π	[m-r]
RL	L	structure half-period	[m]
a	a	maximum beam size	[m]
delV	ΔV	voltage gain	[V]
eta	η	quadrupole packing fraction	

With this information available HILDA uses the procedure KenVar to calculate the following quantities.

m_0	=	931.50	atomic mass unit	[MeV]
τ_l	=	Q/N	charge per beamlet	[C]
$\beta\gamma$	=	$\text{sqrt}((qV)/(m_0A))^2 + 2 \cdot (qV)/(m_0A)$		[]
γ	=	$\text{sqrt}((\beta\gamma)^2 + 1)$		[]
β	=	$(\beta\gamma)/\gamma$		[]
$B\rho$	=	$3.107 \cdot (\beta\gamma) \cdot A/q$	magnetic rigidity	[T-m]

HILDA then uses this information to solve equations pertaining to the beam dynamics. The basic equations used are summarized below. We define the quantities:

G_B	=	B'	Magnetic quad. gradient	[T/m]
G_E	=	E'/v	Electric quad. gradient	[V/m/s]
G	=	G_B or G_E	Quad. gradient	
$\cos(\sigma_0)$	=	$1 - [\eta^2 (3 - 2\eta) (G/B\rho)^2 L^4] / 6$	undepressed tune	

We note that the quantities σ_0 , η , L , are input parameters and $B\rho$ has been calculated. This the above formula for $\cos(\sigma_0)$ determines the quadrupole strength needed to focus the beam.

HILDA now solves for two quantities that we call α and a_m . These are related to the perveance K as will be shown below. The quantity a_m is an average beam radius. The transcendental equation that determines α is shown below.

$$\begin{aligned}\cos(\sigma) &= 1 - \sinh^2(\alpha) [(1 - \cos(\sigma_0)) / \alpha^2 - 2] \\ \sin(\sigma) &= (2L\epsilon_n / \beta\gamma) (\sinh(\alpha)/\alpha) \cdot [\cosh(\alpha) + (\sinh(\alpha)/\alpha) \cdot \sqrt{(1 - \cos(\sigma_0)) / 2}] \cdot (1/a^2) \cdot [1 - 0.12 \eta (1 - \cos(\sigma_0))]^2 \\ f(\alpha) &= \cos(\sigma)^2 + \sin(\sigma)^2 - 1\end{aligned}$$

HILDA finds α by solving $f(\alpha) = 0$. Once this quantity is known the quantities a_m and σ are found from the formulas below.

$$\begin{aligned}a_m^2 &= (2L\epsilon_n / \beta\gamma) \cdot [(\cosh(\alpha) \cdot \sinh(\alpha)) / \alpha - (1 - \cos(\sigma_0)) \cdot \sinh(\alpha) \sinh^2(\alpha/2) / \alpha^3] / \sin(\sigma) \\ \sigma &= \sin^{-1}(\sin(\sigma))\end{aligned}$$

Now that a_m and α are known it is possible to obtain the perveance K and the current I from

$$\begin{aligned}K &= (\alpha \cdot a_m / L)^2 \\ I &= (\beta\gamma)^2 \cdot B\rho \cdot K / 2\end{aligned}\quad [A]$$

The pulse length τ , average voltage gradient E , and the volt seconds V_s that the induction modules must furnish follow from the equations that are shown below.

$$\begin{aligned}\tau &= (\tau I) / I & [s] \\ E &= \Delta V / L & [V/m] \\ V_s &= E \cdot L \cdot t & [V-s]\end{aligned}$$

At this point HILDA has, in principal, solved the beam transport problem. However, it may be that the parameters selected for this station do not yield a solution. For example, there may be no solution for the transcendental equation defining α , or the depressed tune σ becomes less than the allowed lower limit.

After a solution to the beam transport problem is found, HILDA designs a quadrupole. If a solution is not found HILDA terminates the calculation and gives the user an opportunity to try a new parameter set. If a parameter space is being scanned, HILDA goes to the next point in that space.

C. ACCELERATION-TRANSPORT ELEMENTS

The present version of HILDA has modules that design either a superconducting quadrupole or an iron aided pulsed quadrupole. The models for these elements are described in this section.

Iron Aided Pulsed Quadrupole

This element is designed by the HILDA module FeQ20. It is assumed that the beam transport problem has been solved. This means that HILDA has values for the quantities listed below:

<u>Name</u>	<u>Description</u>	<u>Units</u>
a	maximum beam edge radius	[m]
L	transport structure half-period	[m]
η	quadrupole packing fraction	
σ_0	undepressed tune, no space charge	[r]
N	the number of beams	
Bp	magnetic rigidity	[T-m]

HILDA uses these, along with design data from the data file associated with FeQ20, to design a quadrupole array package that will transport the N beamlets. The module writes the design parameters to a file for later processing and returns as its primary output the two quantities shown in the table below.

<u>Name</u>	<u>Description</u>	<u>Units</u>
doTArray	cost of the array	[\$]
roTArray	outer radius of the quadrupole array	[m]

The radial size of the array is needed in the design of the acceleration cell, since the cell must be able to accommodate the quadrupole array.

A picture of this quadrupole is shown above in Figure 2-12, *FeQ20.DAT*. This figure is in the section *The Module Data Files* of the 3MV example. The discussion below will refer to quantities that are in that picture. The correspondence is as show below.

<u>Figure 2-12</u>	<u>Equations</u>	<u>Description</u>	<u>Units</u>
a	a	maximum beam radius	[m]
Raperture	R	aperture radius	[m]
delRPipe	Δr_p	pipe thickness	[m]
delRGap	Δr_g	gap thickness	[m]
delrWire	Δr_w	wire layer thickness	[m]
rWire	r_w	radius to wire center	[m]
delrFe	Δr_{Fe}	yoke thickness	[m]
rFe	r_{Fe}	magnet radius	[m]
	l_o	overhang length	[m]

zlMag	l_m	magnet length	[m]
	B_{max}	maximum wire field	[T]
	B'	field gradient	[T/m]
RL	L	transport section half-period	[m]
eta	η	quadrupole packing fraction	

The module FeQ20 first calculates the beam radius a , the pipe thickness, and the gap thickness as shown below.

$$\begin{aligned} R &= 1.25 \cdot a + .01 \\ \Delta r_p &= 0.03 \cdot R \\ \Delta r_g &= 0.10 \cdot R \end{aligned}$$

We should note that the constants that appear above are parameter data that FeQ20 reads from its data file. Thus, they can be easily changed to other values, should that be desirable. We should also note that there is a data parameter that specifies the minimum pipe thickness and the minimum gap thickness. If the calculated values are less than these minimum, the minimum allowed values are used.

Next the field gradient needed to transport the beam, the wire radius, and the wire field are calculated as shown below.

$$\begin{aligned} B' &= \sqrt{6 \cdot (1 - \cos(\sigma_0)) / (3 - 2\eta)} \cdot B\rho / (L^2 \cdot \eta) \\ \Delta r_w &= 0.005 \cdot (B' r_w / 1.0) \\ r_w &= R + \Delta r_p + \Delta r_g + \Delta r_w / 2 \\ B_{wire} &= B' \cdot r_w \end{aligned}$$

Again we point out that the scaling parameters used in the wire thickness formula are data input parameters. The quantity B_{wire} is checked to be sure that it does not exceed the limit B_{max} ; if it does HILDA terminates the design at this point in the parameter space.

Next are calculated the magnet overhang and the magnet length as shown below.

$$\begin{aligned} l_o &= 0.75 \cdot r_w \\ l_m &= \eta L + 2 l_o \end{aligned}$$

Following this FeQ20 determines the iron thickness, the outer iron radius, and the pitch.

$$\begin{aligned} \Delta r_{Fe} &= B' \cdot r_w^2 / (2 \cdot B_{max}) \\ r_{Fe} &= r_w + \Delta r_w / 2 + \Delta r_{Fe} \\ p &= 2 \cdot r_{Fe} + 0.00 \end{aligned}$$

This latter quantity, pitch, is the beam-center to beam-center spacing when there is more than one beamlet. This is measured horizontally, and is not the diagonal spacing of the beams. This quantity is used to determine the outer radius of quadrupole array assembly. The present version of HILDA assumes that the array is square. This is true if the number of beams is a perfect square. A special formula is furnished for the case of 21 beams.

We note that the yoke thickness Δr_{Fe} is scaled from B_{max} , which is taken to be $1.5T$. In principal the yoke is thick enough to contain the return flux. The cost of the quadrupole is based on the volume of Fe that is used. The quadrupole designed here has a cylindrical yoke with thickness Δr_{Fe} and an inner radius r_{Fe} . It does not contain Fe material in the corners and for this reason the cost of these quadrupole arrays will be less than if the yoke were rectangular in cross-section.

The volume, weight, and cost of one quadrupole are calculated as

$$vol_{Fe1Quad} = \pi \cdot (2 \cdot r_{Fe} - \Delta r_{Fe}) \cdot \Delta r_{Fe} \cdot l_m \quad [m^3]$$

$$wt_{Fe1Quad} = vol_{Fe1Quad} \cdot den_{Fe} \quad [kg]$$

$$cost1Quad = wt_{Fe1Quad} \cdot cCost \cdot qCost \cdot uCost \quad [\$]$$

The weight and cost of the quadrupole assembly for N beams are then calculated.

$$wt_{FeQuad} = wt_{Fe1Quad} \cdot N \quad [kg]$$

$$dolTArray = cost1Quad \cdot N \quad [\$]$$

In the cost calculation there are three cost factors. The factor $uCost$ is the basic cost of the material. The factor $qCost$ is used to adjust that cost for quantity orders. The parameter $cCost$ refers to a complexity factor. In the examples in this report the factors $qCost$ and $cCost$ have been set to 1.0. These three items appear in the FeQ20 data set and can be easily set to appropriate values. The density den_{Fe} is also there.

Upon completion of the quadrupole array design, FeQ20 writes out the complete design data set and the complete set of design parameters. This file is later recovered and included in the logfile for the minimum-cost design.

Superconducting Quadrupole

This element is designed by the HILDA module ScQ30. It is assumed that the beam transport problem has been solved. Thus HILDA has values for the quantities

<u>Name</u>	<u>Description</u>	<u>Units</u>
a	maximum beam edge radius	[m]
L	transport structure half-period	[m]
η	quadrupole pacing fraction	
σ_0	undepressed tune, no space charge	[r]
N	the number of beams	
Bp	magnetic rigidity	[T-m]

HILDA uses these, along with design data from the data file associated with ScQ30 to design a quadrupole array package that will transport the N beamlets. The module writes the design parameters to a file for later processing and returns as its primary output the two quantities shown in the table below.

<u>Name</u>	<u>Description</u>	
dolTArray	cost of the array	[\$]
roTArray	outer radius of the quadrupole array	[m]

The radial size of the array is needed in the design of the acceleration cell, since the cell must be able to accommodate the quadrupole array.

A picture of this quadrupole is shown above in Figure 2-13, *ScQ30.DAT*. This figure is in the section *The Module Data Files* of the 3MV example. There are three pictures in that figure: the first is a transverse view on one SC quadrupole; the second is an end view of the single SC quadrupole; the third is an end view of an array that contains four SC quadrupoles. In the discussion that follows we assume that the reader will refer to those pictures.

We give below a correspondence between: the labels in those figures, the HILDA variables, and the notation used in this section.

<u>HILDA</u>	<u>Equations</u>	<u>Description</u>	<u>Units</u>
eta	η	quadrupole packing fraction	
RL	L	transport structure half-period	[m]
rWire	R_{wi}	inner wire radius	[m]
rWo	R_{wo}	outer wire radius	[m]
drWire	Δ_w	wire thickness	[m]
drPipe	Δ_p	pipe thickness	[m]
drInsul	Δ_{in}	insulation layer thickness	[m]
drCool	Δ_{co}	cooling layer thickness	[m]
BWo	B_{wo}	field at outer wire radius	[T]
Bwire	B_{wi}	field at inner wire radius	[T]
avJ	$\langle J \rangle$	average current density at outer wire radius	[A/m ³]
cJ	J_c	critical current density	[A/m ³]
drWrap	Δ_{wrap}	quadrupole stress wrap thickness	[m]
wQElem	Q_w	Transverse width of the SC Quadrupole package	[m]
pitch	p	beam-center to beam-center spacing	[m]
zlMag	z_{quad}	physical length of the quadrupole package	[m]

The module ScQ30 first calculates: the inner radius R of the beam vacuum pipe, the quadrupole field gradient B', the inner radius R_{wi} of the quadrupole windings, and the magnetic field strength B_{wi} at the inner wire radius. The table below gives the formulas used.

$$\begin{aligned}
 R &= 1.25 \cdot a + .01 \\
 B' &= \sqrt{6 \cdot (1 - \cos(\sigma_0)) / (3 - 2 \cdot \eta)} \cdot B_p / (L^2 \cdot \eta) \\
 R_{wi} &= R + 2 \cdot \Delta_p + 3 \cdot \Delta_{in} + 2 \cdot \Delta_{co} \\
 B_{wi} &= B' \cdot R_{wi}
 \end{aligned}$$

We note that for R and R_{wi} the numerical values of the parameters are data that is input from the ScQ30 data file. Thus, the values can be changed by editing the data in that file. We also note that the values of the quantities Δ_p , Δ_{in} , and Δ_{co} are also read as parameter data from this file.

The bore size is checked to insure that the magnet bore radius to magnetic length is not too large.

$$R_{wi} < 0.5 \cdot \eta L$$

If this limit is not observed, HILDA terminates the quadrupole design and goes to the next point in the parameter space.

The module ScQ30 next solves for the outer wire radius R_{wO} , the average current density $\langle J \rangle$ at the outer wire radius, and the field B_{wO} at the outer wire radius. The following equations are solved to obtain these quantities.

$$\begin{aligned}
 B_{wO} &= B' \cdot R_{wO} & [T] \\
 J_c &= 2.9 \cdot 10^9 \cdot (10.0 - B_{wO}) / 5.0 & [A/m^3] \\
 \langle J \rangle &= \lambda \cdot J_c & [A/m^3] \\
 \mu_0 &= 4 \cdot \pi \cdot 10^{-7} & [Tm^2/A] \\
 f(R_{wO}) &= \ln(R_{wO}/R_{wi}) + (1 - (R_{wi}/R_{wO})^4) / 4 - (2 \cdot B') / (\mu_0 \cdot \langle J \rangle) & [] \\
 f(R_{wO}) &= 0 & [] \\
 B_{wO} &< B_{max} & [T]
 \end{aligned}$$

The coefficients that appear in the above equations are read as data from the ScQ30 data file, see Figure 2-13. For reference to that data file we give in the table below the correspondence between the numerical value, or variable name, and the data name in that file.

<u>ScQ30.DAT</u>	<u>Equation value</u>	<u>Description</u>	<u>Units</u>
cR1 =	1.25	aperture beam radius coefficient	[]
cR2 =	.01	aperture radial clearance	[m]
nPipe =	2	# of pipes	
nInsul =	3	# of insulation layers	
nCool =	2	# of cooling layers	
rWtRL =	0.50	bore size limit factor	[]
cJCoeff =	$2.9 \cdot 10^9$	slope of critical current density cJ curve	[A/m ³]
cJBnj =	10.0	numerator field parameter	[T]
cJBdj =	5.0	denominator field parameter	[T]
rLamda =	λ	SC wire packing fraction	
BWomax =	B_{max}	B at outer wire, maximum	[T]

A check is done to insure that the wire thickness is not too thin, or too thick, and that the field B_{wO} at the outer radius is greater than 0 and less than the maximum B_{max} . These limits are furnished as data in the ScQ30 data file. If for some reason ScQ30 cannot find a solution that satisfies these equations and meets the limit criteria, the design of the SC quadrupole is terminated and HILDA goes to the next point in the parameter space.

When a solution is obtained the outer wrap Δ_{wrap} that encloses the quadrupole windings, the horizontal width Q_w of the SC quadrupole package, and the beam-center to beam-center spacing (pitch) p are calculated as shown below.

$$\Delta_{wrap} = 0.01 \cdot (B_{w0}/5.0)^2 \cdot (r_{w0}/0.12)$$

$$Q_w = 2 \cdot (R_{w0} + \Delta_{wrap} + 0.0015)$$

$$p = Q_w$$

The square quadrupole modules are stacked together to form a package, that we refer to as the quadrupole array. This array contains N quadrupoles, where N is the number of beamlets. We assume here that N is a perfect square, e.g., 1, 2, 4, 16, This array structure is shown in the third picture of Figure 2-13 that we previously referred to. The outer radius of the quadrupole array is obtained from the formula below.

$$roTArray = p \cdot \sqrt{2} \cdot (2 \cdot 0.25 + \sqrt{N}) / 2 \quad [m]$$

The length of the quadrupoles, and thus of the array, is determined and a check is done to insure that it will fit in the allotted space.

$$z_{quad} = \eta L + 2 \cdot (R_{w0} + 0.00)$$

$$z_{quad} < L \cdot (1 - 0.10)$$

The coefficients that appear in the above equations are read as data from the ScQ30 data file. For reference to that data file we give in the table below the correspondence between the numerical value and the data name in that file.

<u>ScQ30.DAT</u>	<u>Equation value</u>	<u>Description</u>	<u>Units</u>
sWrap =	0.01	wrap thickness used for BWrap, rWrap	[m]
BWrap =	5.0	field used for scaling the Quad. Wrap	[T]
rWrap =	0.12	radius used with BWrap	[m]
dShell =	0.0015	thickness of the outer shell	[m]
fCollar =	0.25	array wrap (collar) width, fraction of pitch	[]
zQend =	0.00	space used by quadrupole end packaging	[m]
fZSpace =	0.10	no-magnet space limit, fraction of half-period L	[m]

At this point in the design the quadrupole array has been completely determined. The rest of the module is devoted to finding the cost of the array. This cost is determined by finding the amount of material and then multiplying that by the cost per unit of material. A look at the pictures in Figure 2-13 will show that it is now necessary for ScQ30 to determine the volumes of the various components. This is a simple, though rather tedious, calculation. In the discussion that follows we shall sketch out what is calculated.

The module ScQ30 first does the calculations for one quadrupole. All the quadrupoles in the array are assumed to be the same. It calculates the cross-sectional areas of the:

- quadrupole package
- stress wrap around the windings, includes the shell
- cable, includes the SC and the non-SC cable material
- beam pipe, inside of the vacuum pipe
- pipes, insulation and cooling layers

These components can be easily identified by looking at the second picture in Figure 2-13.

Next, these cross-sectional areas are multiplied by the quadrupole package length z_{quad} to obtain the corresponding volumes. Once the volumes are known ScQ30 proceeds to obtain the weights of the components:

- SC cable, using the packing factor λ
- non-SC cable material
- conductor material, SC plus non-SC
- outer SC quadrupole wrap, includes the outer shell as same as the stress wrap
- pipe layers
- insulation layers
- cooling layers
- total quadrupole package, includes all the above components

The costs are calculated using by multiplying the weights by three quantities: the unit cost [\$/kg] of the material, a quantity factor, and a complexity factor. The values for these parameters are furnished in the ScQ30 data file.

Note that the present version of ScQ30 does not use the costs of the pipe, insulation, and insulation materials. Instead it uses a quantity called BFactor in the data set to calculate a cost of everything except the cable and the wrap. This is done as shown below.

$$\text{\$balance} = \text{BFactor} \cdot (\text{\$cable} + \text{\$wrap})$$

$$\text{\$quad} = \text{\$cable} + \text{\$wrap} + \text{\$balance}$$

At this point in the calculation ScQ30 has obtained the cost of one of the N quadrupole channels that make up the quadrupole array. These channels are stacked together to form the quadrupole array; this is shown in the third picture in Figure 2-13.

Cross-sectional areas are determined for the:

- SC quadrupole channels
- beam pipes, vacuum cross-sectional area
- outer array collar
- total array of N quadrupoles, includes the outer wrap

These areas are then multiplied by the length z_{quad} to obtain the corresponding volumes. For the quadrupole array of N channels the following weights are calculated.

- SC cable
- non-SC cable
- outer quadrupole stress wraps
- pipes
- insulation layers
- cooling layers
- quadrupole channels, no outer array collar
- outer array collar
- total array assembly

The array component dollar costs are obtained by multiplying the array channel component costs by the number of channels N . The outer collar cost is calculated by multiplying the amount of material in the collar by the unit cost [\$/kg], the quantity factor, and the complexity factor. The total cost of the array is obtained by adding the array outer wrap cost to the cost of N quadrupole channels and then multiplying that cost by an array complexity factor. The values of the needed parameters are obtained when ScQ30 reads the data file ScQ30.DAT. Examples of those files are found in the 3MV and 3000MV examples furnished in this report.

Once the cost calculation is finished the complete set of design parameters and all component areas, volumes, weights, and costs are written to a file. This information can then be recovered and included in the logfile that HILDA writes for the minimum-cost design.

The module ScQ30 is now finished with its task of designing and costing an SC quadrupole array. Upon exit it returns the cost of the array, dolTArray , and the outer radius of the array, roTArray .

Acceleration Modules

In the acceleration-transport section of the heavy ion fusion driver HILDA designs focusing elements and acceleration cells. This section describes the model for the module StrucCore that designs the acceleration cells. There are two pictures of this structure in the Figure 2-14, *StrucCore.DAT*, which is in the section *StrucCore.DAT Acceleration Cell Design Data* of the 3MV example previously discussed in this report. Refer to those pictures to understand the discussion that follows.

StrucCore designs an acceleration cell that contains induction cores that are concentric with the beam center line. The cell that the modules are packaged in consists of an outer cell housing, end plates, and an inner bobbin. A gap insulator, concentric with the beam, is part of the cell inner bobbin. Power leads are brought in through the outer cell housing to furnish power to the cores. The components are separated by insulation material that is part of the core modules and the cell housing. The core modules are wound from amorphous material; the material's flux swing and the required volt-seconds determine the core cross-section areas. The first picture in Figure 2-14 shows an example of a cell with two induction cores. The module StrucCore can stack as many cores as are necessary. The present version of StrucCore stacks the cores side-by-side; it can not stack the cores vertically.

The second picture in that figure shows a cell that is made up of exactly one core module. The labels in that picture are the names of the corresponding variables in StrucCore. In the discussion that follows we outline the algorithm that StrucCore uses. The design of the acceleration cell is principally a task of furnishing the needed amount of amorphous core material, while at the same time meeting specified design constraints.

The acceleration design module StrucCore is invoked only when HILDA has completed a successful design of the quadrupoles. This means that there are available the quantities shown below. We use the notation and units that are used in the StrucCore module.

<u>Name</u>	<u>Description</u>	<u>Beam dynamics Name</u>	<u>Units</u>
Vs	volt-sec acceleration per half period	V_s	[V-s]
nB	number of beamlets	N	
roA	outer radius of the quadrupole array		[cm]
VG	voltage gain per half-period at this station	ΔV	[kV]
HLP	lattice half-period length	L	[cm]

Upon completion of the design of the acceleration cell the following quantities are returned.

<u>Name</u>	<u>Description</u>	<u>Units</u>
dolAStruct	total cost of the acceleration cell, including quadrupoles	[\$]

StrucCore has been developed from notes and information furnished by C. Fong of LBL. The design of the cell proceeds in a number of steps, which we outline in the discussion that follows. After a minimum-cost design has been obtained a logfile *StrucCore.OUT* is written. This file contains the quantities which completely define the acceleration structure; all the design data and all the calculated quantities are in that file. It is later included in the logfile that HILDA returns after finding a minimum-cost design.

The StrucCore design steps are presented below in short sections that are appropriately labeled.

Material Data

Material properties are obtained from the *StrucCore.DAT* file. This file contains flags that define the selection of materials and the data for those materials. The file is described above in the sections that present the 3MV example. StrucCore reads data that defines the tape that is used to wind the induction coils. It also needs to know the material that is used to construct the module that houses the core, and the material that is used for the cell housing into which the induction core modules are placed. We note that there is an elasticity modulus scaling factor associated with the cell housing material. StrucCore uses this to determine how thick to make some of the components in the cell.

The components of the core and the cell are separated electrically by insulation material. It is necessary to know the voltage breakdown limit of the insulation material. This information is also needed for the cell gap insulator and the dielectric coolant. We treat the acceleration gap vacuum as a dielectric medium and furnish a voltage breakdown limit for it.

All the materials have a density that is used to find the material weight. The unit cost furnishes the cost of the material, once the weight is known. The quantity factor is used to include any price discount that arises from quantity orders of the material. The complexity factor can be used to adjust the price of the material. This could represent a special material cost, or it could be used to take into account the assembly of the components that contain the material, or the intricacy of the part itself.

The unit costs shown here were derived by estimating the cost of accelerator piece parts at driver quantities. This was done for typical cell parts with assistance from industrial suppliers. Also considered in addition to the mill run cost of materials were the costs of labor, overhead, profit, and amortized tooling . Here, for a full scale driver, the quantity and complexity factors are set at 1.0.

These parameters, along with representative values taken from the 3MV example discussed previously in this report, are shown below.

Core Amorphous Tape Material

Metglas 2605 S2 - wound and annealed

denAm	7.1800	density	[gm/cm ³]
cFacAm	1	complexity factor	[]
qFacAm	1	quantity factor	[]
uCostAm	5.0	unit cost	[\$/kg]
pFAm	0.80	radial packing fraction	[]
delBAm	2.5	maximum flux swing of material	[T]

Core Tape Width

wTape	20.32	width of amorphous tape	[cm]
aRCmax	4.0	maximum tape winding height to width aspect ratio	[]

Core Submodule Housing Material

Low Carbon steel - welded & machined - 1020

denCSM	8.1657	density	[gm/cm ³]
cFacCSM	1.0	complexity factor	[]
qFacCSM	1.0	quantity factor	[]
uCostCSM	9.8987	unit cost	[\$/kg]

Cell Housing Material

Low Carbon steel - welded & machined - 1020

denCH	8.1657	density	[gm/cm ³]
cFacCH	1.0	complexity factor	[]
qFacCH	1.0	quantity factor	[]
uCostCH	9.8987	unit cost	[\$/kg]
emCH	30	elasticity modulus scaling factor	[10 ⁶ lb/in ²]

Core & Cell Housing Insulation

Polyethylene LP-390-C Dielectric, injection molded

denIN	0.9850	density	[gm/cm ³]
cFacIN	1.0	complexity factor	[]
qFacIN	1.0	quantity factor	[]
uCostIN	20.6417	unit cost	[\$/kg]
bVoltIN	197.00	voltage breakdown limit, dielectric strength	[kV/cm]

Gap Insulator Material

Alumina - pressure cast & brazed large dia. - to 58" by 1" thick

denGI	3.7170	density	[gm/cm ³]
cFacGI	1.0	complexity factor	[]
qFacGI	1.0	quantity factor	[]
uCostGI	428.29	unit cost	[\$/kg]
bVoltGI	12.0	voltage breakdown limit, dielectric strength	[kV/cm]

Dielectric Coolant

Freon

denDiC	1.8	density	[gm/cm ³]
cFacDiC	1.0	complexity factor	[]
qFacDiC	1.0	quantity factor	[]
uCostDiC	1.66	unit cost	[\$/kg]
bVoltDiC	39.00	voltage breakdown limit, dielectric strength	[kV/cm]

Voltage Breakdown Strengths [kV/cm]

Vdss	=	bVoltIN	core & cell housing insulation	[kV/cm]
Vdsl	=	bVoltDiC	radial insulator gap	[kV/cm]
bVRGap	=	50.000	acceleration gap vacuum, voltage breakdown limit	[KV/cm]

Limits and Scaling Factors

The induction cores depend on pulse forming networks that supply the required voltage to the cores. There is a limit to the amount of voltage that can be supplied by the individual PFNs. Space must be left for the support of the quadrupoles and for other ancillary items. The acceleration cell cannot occupy the complete half period length. The gap insulator cannot be too long; it must be somewhat shorter than the cell. The cell housing and the core housing (power lead) cannot be too thin. The thickness of the core housing top, the core housing bottom, and the cell housing top support ring are all obtained by scaling the housing thicknesses.

The modulus of elasticity of material used for the housings is referenced to aluminum and the formula in StrucCore uses 10 [10⁶lb/in²] as the reference for which the scaling factor is 1.0.

The insulation that separates the cell and core components must not be too thin. If the required voltage causes the calculated values to be less than these limits, the thicknesses are reset to these limits.

The gap insulator thickness must be specified. And the perimeter support factor must be set. Both of these take into account the number of beamlets in the transported beam.

All this parameter data is determined by StrucCore before starting to design the acceleration cell.

These parameters and typical values taken from the 3MV example previously discussed in this report are shown in the tables below.

Maximum PFN Peak Kilovolts per Core

pV	200.0	available PFN peak kilovolts per core	[kV]
----	-------	---------------------------------------	------

Minimum Length for Quadrupole Support

QAL	0.10	z-axis fraction of half-period length	[]
-----	------	---------------------------------------	-----

Maximum Length for Gap Insulator.

gILmax	0.80	z-axis fraction of cell length cellL	[]
--------	------	--------------------------------------	-----

Housing Thicknesses Factors

Average of data for weights of 5000lb & 11000lb

zIH	2.0	cell housing thickness	[cm]
zIPH	1.0	core housing/power lead thickness	[cm]
proC	2	scaling factor for core housing top	[]
priM	2	scaling factor for core housing bottom	[]
proH	2	scaling factor for cell housing top support ring	[]

Elasticity Scaling Factor for Cell Housing Radius

peM	10	referenced to aluminum for scaling other materials	[10 ⁶ lb/in ²]
-----	----	--	---------------------------------------

Minimum Insulation Thicknesses

zICimin	0.200	minimum insulation thickness	[cm]
dRgmin	0.500	minimum radial insulator gap space	[cm]

Insulator Thickness & Perimeter Support Factor

16 beam data

tl	3.0	acc. gap insulator thickness	[cm]
fQ	1.1	quad array perimeter support factor	[]

Acceleration Cell Design

We now proceed to the design of the acceleration-transport cell. This structure is placed around the transport array; usually an array of focussing or defocussing quadrupoles. We note here that in principal it is possible to have only a transport array with no acceleration. StrucCore has been designed so that this can be done by furnishing a voltage gain of zero. When the voltage gain is zero, the cost of the cell is set to zero.

In the tables that follow we show the defining equations and we identify the quantities as they are obtained. Most of what follows should be self-explanatory. The dimensions that are obtained are shown in the second picture of Figure 2-14. The parameters used in the calculations have been previously identified above. We will not comment individually on each of these tables. The headings identify the quantities being calculated.

Core & Cell Thicknesses

delTC	=	proC • zIPH	core housing top	[cm]
delBC	=	priM • zIPH	core housing bottom	[cm]
delCH	=	proH • zIH	cell housing top support ring	[cm]
delCH	=	delCH • sqrt(peM/emCH)	resize if material that isn't aluminum	[cm]

Limits on Support and Gap Insulator Lengths

QAlmin	=	QAl • HLP	quad array perimeter support factor	[cm]
--------	---	-----------	-------------------------------------	------

Number of Cores Needed

The number of cores, numCore, needed to furnish the required voltage gain depends on the available peak PFN voltage. This number must satisfy the relation below.

$$\text{numCore} \cdot pV \geq VG.$$

Insulation Width, Radial Insulator Clearance

$$zICi = \max (pV/Vdss, zICimin) \quad [cm]$$

$$delRg = \max (pV/Vdsl, dRgmin) \quad [cm]$$

Cell Length and Quadrupole Support Length

$$cellL = 2 \cdot zIH + numCore \cdot (3 \cdot zICi + 2 \cdot zIPH + wTape) \quad [cm]$$

$$QAI = HLP - cellL \quad [cm]$$

Check on Cell Length

$$QAI \leq HLP - cellL \quad [cm]$$

Set Maximum Gap Insulator Length

$$gILmax = gILmax \cdot cellL \quad [cm]$$

Determine Gap Insulator Length

$$gIL = \max (numCore \cdot pV / bVRGap, numCore \cdot pV / bVoltGI) \quad [cm]$$

Check on Gap Insulator Length

$$gIL \leq gILmax \quad [cm]$$

Size the Core

The cross-sectional area of the induction core is determined by the required volt-seconds. The volt-seconds was calculated in the beam transport section of HILDA. We recall that this was obtained as $V_s = E \cdot L \cdot \tau$; (electric field • half-period • beam pulse length). The data file StrucCore.DAT furnishes the flux swing for the core material. In principal this is the maximum flux swing. However, we note that if we wish to have constant size cores we will have to set the flux swing to a value that is probably other than the maximum. Future versions of this module should incorporate a way of specifying the desired core size. The core area and the tape width determine the radial size (height) of the core. There is a limit to the height/width ratio; too high and the core is not stable. This ratio is determined here and subsequently checked against that limit. The calculations are shown below.

$$acAM = Vs / delBAm \cdot 1000 \quad \text{core amor. mat. total cross-sec. area} \quad [cm^2]$$

$$aC = acAM / numCore / pFAM \quad \text{core module cross-sec. area} \quad [cm^2]$$

$$hC = aC / wTape \quad \text{core height} \quad [cm]$$

$$aRC = hC / wTape \quad \text{core, height to width ratio} \quad []$$

Check that the Core Winding Radius

$$aRC \leq aRCMax \quad [cm]$$

Determine Housing and Core Radii

riH	=	fQ • roA	cell housing inner radius	[cm]
riM	=	riH + tI + delRg	core housing inner radius	[cm]
riC	=	riM + delBC	core material inner radius	[cm]
roC	=	riC + hC	core material outer radius	[cm]
roCore	=	roC + delTC + zICi	core housing outer radius	[cm]
roHSG	=	roCore + delCH	cell housing outer radius	[cm]

Weights of Core and Cell Materials

The geometric dimensions of the core and cell are known; it is now possible to calculate the material weights. This is done by multiplying the volume of the component times the density of the material. Note that the amorphous material does not occupy all the core module space; the parameter pFAM determines the fraction of volume that is amorphous material. We use $PI = \pi = 3.14592654$ in the following equations

Core Material Weights

wtCore	=	$PI \cdot (roC + riC) \cdot (roC - riC) \cdot wTape \cdot denAm \cdot pFAM$	amorphous material per core module	[gm]
wtAm	=	numCore • wtCore	HLP (cell) core Amorphous material	[gm]

Core Submodule Weights

wtCEP	=	$PI \cdot (roC + zICi + riC) \cdot (roC + zICi - riC) \cdot zLPH \cdot denCSM$	Core Submodule Housing/power lead weight (end plate) There are 2 end plates per core housing	[gm]
wtCOH	=	$PI \cdot (2 \cdot (roC + zICi) + delTC) \cdot delTC \cdot (2 \cdot zLPH + 2 \cdot zICi + wTape) \cdot denCSM$	Core Submodule outer housing weight	[gm]
wtCOB	=	$PI \cdot (2 \cdot riM + delBC) \cdot delBC \cdot (2 \cdot zLPH + 2 \cdot zICi + wTape) \cdot denCSM$	Core Submodule inner bobbin weight	[gm]
wtCHPC	=	$2 \cdot wtCEP + wtCOH + wtCOB$	Core housing weight per core	[gm]
wtCorH	=	numCore • wtCHPC	HLP weight of the core submodule housings	[gm]

Insulation Weights for Core & Cell Housing

$wt_{CEI} = \frac{PI \cdot (2 \cdot ri_C + h_C) \cdot h_C \cdot z_{lCi}}{den_{IN}}$	Core end plate insulation There are 2 end plates per core housing	[gm]
$wt_{COI} = \frac{PI \cdot (2 \cdot ro_C + z_{lCi}) \cdot z_{lCi} \cdot (2 \cdot z_{lCi} + w_{Tape})}{den_{IN}}$	Core outer housing insulation	[gm]
$wt_{CI} = 2 \cdot wt_{CEI} + wt_{COI}$	Core insulation weight	[gm]
$wt_{CHIE} = \frac{PI \cdot (ro_{Core} + (ri_M - z_{lCi})) \cdot (ro_{Core} - (ri_M - z_{lCi})) \cdot z_{lCi}}{den_{In}}$	Cell housing end insulation per core module	[gm]
$wt_{CHI} = wt_{CHIE}$	Cell housing insulation weight	[gm]
$wt_{CCI} = num_{Core} \cdot (wt_{CI} + wt_{CHI})$	HLP core/cell housing insulation Note: the end plate weight must be added to the end of the periodic cell structure to close the structure	[gm]

Cell Housing Weights

$wt_{HSGEP} = \frac{2 \cdot (PI \cdot (ro_{HSG} + ri_H) \cdot (ro_{HSG} - ri_H) \cdot z_{lH})}{den_{CH}}$	Cell housing end plates/pair	[gm]
$wt_{OHR} = \frac{PI \cdot (ro_{HSG} + (ro_{Core})) \cdot (ro_{HSG} - (ro_{Core})) \cdot (cellL - 2 \cdot z_{lH})}{den_{CH}}$	Cell housing support ring	[gm]
$wt_{HSGGI} = \frac{PI \cdot (2 \cdot ri_H + z_{lH}) \cdot z_{lH} \cdot (cellL - 2 \cdot z_{lH} - g_{IL})}{den_{CH}}$	Gap insulator support ring	[gm]
$wt_{CelH} = wt_{HSGEP} + wt_{OHR} + wt_{HSGGI}$	HLP cell housing	[gm]
$wt_{GI} = \frac{PI \cdot (2 \cdot ri_H + t_l) \cdot t_l \cdot g_{IL}}{den_{GI}}$	HLP gap insulator	[gm]
$wt_{DiC} = \frac{(PI \cdot (ri_M + (ri_H)) \cdot (ri_M - (ri_H)) \cdot (cellL - 2 \cdot z_{lH}) - wt_{HSGGI}/den_{CH} - wt_{GI}/den_{GI})}{den_{DiC}}$	HLP electric coolant	[gm]

HLP Component Weights in Kilograms

$wt_{Am} = 1000 \cdot wt_{Am}$	amorphous core	[kg]
$wt_{CorH} = 1000 \cdot wt_{CorH}$	submodule housing/power leads	[kg]
$wt_{CelH} = 1000 \cdot wt_{CelH}$	cell housing	[kg]
$wt_{CCI} = 1000 \cdot wt_{CCI}$	core housing insulators	[kg]
$wt_{GI} = 1000 \cdot wt_{GI}$	gap insulator	[kg]
$wt_{DiC} = 1000 \cdot wt_{DiC}$	dielectric coolant	[kg]

HLP Component Costs

The dollar cost of the components is obtained by multiplying the weight of the material by the unit cost. This result is multiplied by the quantity factor and the complexity factor to obtain the final costs of the component. In the examples presented in this report these factors are set to 1.

dolAm	=	$wtAm \cdot cFacAm \cdot qFacAm \cdot uCostAm$	amorphous core material	[\$]
dolCorH	=	$wtCorH \cdot cFacCSM \cdot qFacCSM \cdot uCostCSM$	core housing/power lead material	[\$]
dolCelH	=	$wtCelH \cdot cFacCH \cdot qFacCH \cdot uCostCH$	cell housing material	[\$]
dolCCI	=	$wtCCI \cdot cFacIN \cdot qFacIN \cdot uCostIN$	core & cell housing insulation	[\$]
dolGI	=	$wtGI \cdot cFacGI \cdot qFacGI \cdot uCostGI$	gap insulator material	[\$]
dolDiC	=	$wtDiC \cdot cFacDiC \cdot qFacDiC \cdot uCostDiC$	dielectric coolant	[\$]

Total Cell Weight and Cost per HLP

The final task in the calculation is to get the total cost and the total weight of the acceleration cell. This is done by adding up the component costs.

wtAStruct	=	$wtAm + wtCorH + wtCelH + wtCCI + wtGI + wtDiC$	module weight	[kg]
dolAStruct	=	$dolAm + dolCorH + dolCelH + dolCCI + dolGI + dolDiC$	module cost	[\$]

The design and costing of the acceleration-transport cell is complete. StrucCore writes a file that contains all the design parameters, all the calculated dimensions, all the component weights, and all the component costs. This file can later be retrieved and included in the logfile that HILDA produces for a minimum-cost design.

We note here that this version of StrucCore does not include the cost of the pulse forming networks, nor does it include any cost associated with power losses in the induction cores. We also note that the volt-seconds furnished to StrucCore does not include any extended modeling of the rise and fall of the real pulse. These facts should be included in future versions of this module.

We also recall that the modules are stacked longitudinally. There is no provision for radial stacking. We again comment that there is no direct way to keep the core sizes constant, while staying within material flux swing limits. Future versions of StrucCore should include these capabilities.

4. MAINTAINING HILDA

A. INSTALLATION

How to install HILDA in a computing environment is discussed in this section. The Hilda Program Document consists of this report and additionally the contents of the associated floppy disks. The installation of HILDA consists of transferring the appropriate files from these disks into the current computer environment. After this transfer the resident FORTRAN compiler can be invoked to create an executable image. Once the HILDA executable image has been created and the necessary data sets have been supplied, the program can be run.

How the initial file transfer and data setup are done depends on the user's computing environment. The Microsoft Word *ReadMe* document in the folder *MSW/HILDA/VAX* on the Hilda Program Document floppy disks describes in detail the installation of HILDA. This document gives instructions on how to install HILDA for users who have access to the LBL VAX cluster. It contains the VAX command files for those that do not have such access, but do run on a VAX and wish to install HILDA in that computing environment. It describes how to transfer (download or copy) the HILDA files from the Hilda Program Document disks into any computing environment in which FORTRAN is available. This information is not explicitly repeated here, since it is available in the aforementioned folder. As we have pointed out, the Hilda Program Document is not just this report but also includes the complete contents of the associated floppy disks. More information can be found in the *APPENDIX* of this report.

B. UPDATING CONVENTIONS AND PHILOSOPHY

Updating HILDA is not a difficult task. However, it is a task that should be taken seriously. We mention this because it is very easy to casually place updates into HILDA and in the process of doing this nullify the effort that has been put into the documentation of HILDA. The present version of HILDA has been built in a very specific way. We have consistently written the HILDA modules using the *template* that is furnished with this HPD in the folder *MSW/Hilda/ALL* of the HPD floppy disks. The modules are fully formatted and are meant to serve as their own documentation. The modules are then written as flat ASCII files, which are downloaded into the computing environment. Using simple drivers the modules are pre-tested before being included into the HILDA program. Some examples of these drivers and associated source code are included in the folder *MSW/Hilda/Dnn*. By adhering to this convention we end up with the documentation for the modules at the time they are written. We also insure that the modules that are in the HILDA document are modules that will execute. The pre-testing allows us to have some experience with the modules and thus determine that they carry out their tasks appropriately; before installing them in HILDA. When they are placed into the program they become black boxes that simply take input and furnish output.

We do not explain here how to update HILDA. This updating should be done by someone who is experienced in FORTRAN and who will consistently follow the above mentioned convention. The program is very modular and can easily accommodate other modules. However, it is absolutely necessary to understand the structure of the modules and how they interact in order to add modules successfully, or to change the ones that are there. Instructions on the updating of HILDA can be found in the *ReadMe* files in the HPD disks; see the *APPENDIX; The Hilda Program Document Disk*.

C. FUTURE UPDATES

The most immediate updates for the present version of HILDA deal with the data input that is needed and the run output is generated. The next level of updates will expand the capabilities of HILDA to cost a complete machine. HILDA presently finds a minimum-cost design at user chosen stations along the machine. The costing of a complete machine requires that the station costs be appropriately combined.

Input

HILDA needs a better user interface that decreases the amount of data that a user must process when doing a minimum-cost design for a complete machine. The present version of HILDA has been built to do minimum-cost designs at a particular machine station. The user should be able to set up the data for all the stations in the machine before running HILDA. Presently the module data files must be reset if they are not the same at all machine stations. It is also evident that modules are needed that allow the user to modify the station data during the HILDA run. These modules are rather easy to build and incorporate into HILDA.

Output

After HILDA has generated minimum-cost designs at the selected stations, it becomes necessary to analyze the results. It seems appropriate to use standard programs to do this; e.g., a spreadsheet program. The standard HILDA logfiles presently contain all the design information. What is needed are modules that allow the user to create files that are appropriate for input to analysis programs. It also seems appropriate to keep in mind that in the future it will be possible to interface HILDA directly with such programs. By this we mean that the output from the HILDA can be made available to the analysis program, as HILDA runs. This could prove quite useful in guiding a user to the proper total machine design in which the parameter transition between stations was taken into account. The modular construction of HILDA will accommodate this without disturbing the basic cost algorithms.

Additional Costing Capabilities

A minimum-cost design of a complete machine requires that HILDA be updated to include more components of the design. We list here some of the items that will be added into the costing algorithms.

- pulse forming network costs
- acceleration core power losses
- linear costs such as vacuum, alignment, and refrigeration
- special elements such as: combiners, steering elements, pulse shape correctors








We also should consider including into the costing algorithm an ability to take the individual station costs and use them to obtain a complete machine cost. Presently this total cost can only be obtained from the analysis of the HILDA output produced at each station.

APPENDIX: HILDA PROGRAM DOCUMENT DISKS

A. GUIDE TO THE HPD DISKS

The complete Hilda Program Document consists of this report and the contents of the associated floppy disks. These disks contain more information than is in this report. There are three 1.4Mb Macintosh formatted floppy disks. These are labeled *HPD Disk 1-3*, *HPD Disk 2-3*, and *HPD Disk 3-3*. The contents of the three disks are shown below in Figure 5-1, *HPD Floppy Disks*. The .sit files on these disks are compressed files. They contain self-unstuffers and can be unstuffed by double clicking on the file icons. The two files *Hilda on ILSE.sit* and *Hilda/TXT.sit* become the folders *Hilda on ILSE* and *Hilda/TXT* shown below in Figure 5-2. The files *Hilda/MSW.sit 1* and *Hilda/MSW.sit 2* contain the contents of the folder *Hilda/MSW* shown in Figure 5-2. Readers interested in obtaining the HPD disks can do so by submitting a request to the authors at the Lawrence Berkeley Laboratory, 1 Cyclotron Road, Berkeley, CA, 94720.

Figure 5-1. HPD Floppy Disks

HPD Disk 1-3		HPD Disk 2-3	
5 items 1,215K in disk 201K available		1 item 891K in disk 525K available	
			
ReadME	HPD5	Hilda/MSW.sit 1	
			
	MS Word/		
			
Hilda on ILSE.sit	Hilda/TXT.sit		
		HPD Disk3-3	
		1 item 862K in disk 553K available	
			
		Hilda/MSW.sit 2	

The basic structure of the folders is a *ReadMe* document and the associated folders at that level. Each folder has the same structure. The *ReadMe* document is a Microsoft Word document. The folders that have MSW in their name are fully formatted Microsoft Word documents. Those that have TXT in their name are ASCII flat text files that can be read by any word processor, or text editor, that can read ASCII text files. The basic document is the MS Word file. The text files are the same information, but they are written as text files. These text files can be downloaded into most computing environments.

The HILDA module source files are in the folder *MSW/Hilda/ALL* and the HILDA utility source files are in the folder *MSW/Hilda/U*. In the folder *MSW/Hilda/DATA* there are data files for the examples and the output from these examples. In principle the version of HILDA that is in the source files can read the data files and produce output files that are the same as those included here. The folder *MSW/Hilda/EQU* contains

version files for the HILDA modules. There is a folder for creating HILDA on a VAX, which is *MSW/Hilda/VAX*.

Most users will not be interested in all of this information. Information about the folder contents can be obtained by consulting the ReadMe documents in each of the folders. Below in Figure 5-2, *HPD/folders: ReadME* the top level ReadME file for the HPD folders is shown.


Figure 5-2. HPD/folders: ReadME


Date: January 27, 1992
File: HPD/folders: ReadME
Subject: HILDA Program Document disks


This folder contains the basic HILDA folders as shown in the figure below.


HPD/folders


6 items 128,388K in disk 10,305K



ReadME


HPD


Hilda/MSW


MS Word/


HILDA on ILSE


Hilda/TXT

The contents of these folders make up the Hilda Program Document disk, referred to as the HPD disk. This is really a collection of floppy disks. In each folder is a ReadME file that pertains to the folders at that level. At the current level we have four folders. The *Hilda/MSW* and the *Hilda/TXT* folders contain essentially the same information. The */MSW* folder has fully formatted Microsoft Word files that make up HILDA and the associate HPD examples. The */TXT* folder contains the same information with the files being flat ASCII files. This means that the */MSW* folder files must be read with Microsoft word, or a word processor capable of reading MS Word files. However, the */TXT* files should be readable by any word or text processor that can read ASCII files. This is not necessarily a simply a duplication of information. The MS Word files also contain graphics, pictures that relate to HILDA. No such pictures exist in the */TXT* files. Also, some of the files are not repeated in the */TXT* files. This is because the */MSW* files are considered to be the primary documentation files. The */TXT* files are images that can be down loaded into other computing environments.

The folder *MS Word* contains documents that pertain to the use of the word processor Microsoft Word with the HPD documents.

The folder *HILDA on ILSE* contains files that make up an example that was created by running HILDA on the parameters in the ILSE document, LBL PUB 5219-ILSE, that is sometimes referred to as the brown book ILSE. This example included here as supplementary information. It gives a flavor of what HILDA could easily do if it were suitably interfaced with a spreadsheet program.

90

B. INSTALLING HILDA

We reproduce here in Figure 5-3 the file *HPD/folder:Hilda/MSW:MSW/Hilda/VAX:Meat Me* from the HPD disks. This file tells how to install HILDA in a VAX computer.

Figure 5-3. Hilda Installation

Date: October 7, 1991
File: HPD/folders:Hilda/MSW:MSW/Hilda/VAX:ReadME
Subject: Installation of HILDA

This folder contains VAX command files for the installation and running of HILDA.

Installing a VAX Version of HILDA Using the Files in WEEK:[BENDING.HILDA]

A VAX version of HILDA is available. It is stored on the LBL VAX CSA cluster disks. The following are step by step instructions on how to install HILDA in a VAX directory. Following these steps will install HILDA in the user's directory. We shall assume for this example that the current default directory is [USER]. In actual fact it will be whatever directory your current default directory is.

- Create a directory that will be the root directory for HILDA. In this example we type the command

CREATE/DIRECTORY [USER.HILDA]

- Make this directory your default directory. In this example we would type the command
SET DEFAULT [USER.HILDA]

It is now necessary to know which LBL VAX CSA cluster disk currently contains the HILDA files. This can be accomplished by using the DISKSPACE tool; which runs on the LBL VAX CSA cluster. In this example we assure you have access to this tool.

- Type the command

DISKSPACE BENDING

The output from this command will show an LBL disk logical name on the line that gives the weekly disk usage for the account BENDING. In this example we assume that this is LBL120. If it is some number other than 120, use the currently displayed value.

- Define the logical name for the disk that HILDA is on by typing

DEFINE LBLnnn LBL120

NOTE: use the value displayed by the DISKSPACE tool.

- Put all the HILDA .com files into the [USER.HILDA] directory by typing
COPY LBLnnn:[BENDING.HILDA]*.COM.0 *.*.0

- Edit the just created LOGIN.COM file to:

- replace *rootName* in the statement

\$ HILDARoot := 'rootName

with the correct HILDA root directory name. In this example the statement becomes

\$ HILDARoot := USER.HILDA

- Save this edited file, it is the correct login.com file for HILDA.

Figure 5-3. Hilda Installation (continued)

Instead of editing the LOGIN.COM file you could type

```
rootName := USER.HILDA
```

However, this symbol definition, which is needed, will vanish when you logoff and you would have to reestablish this symbol definition each time you login to run HILDA.

- Execute this VAX command file by typing
@LOGIN.COM

If you have any error messages issued during the execution of the command file, you should check that the above quantities have been correctly entered.

- Verify that all is right before installing HILDA by typing
SHOW SYMBOL HILDARoot
SHOW LOGICAL LBLnnn

The results returned should agree with what you have been instructed to type in the above instructions. Up to this point about the only mistake that you can make it to have incorrectly entered the requested information.

You are now ready to actually install HILDA by creating the directories, transferring the source files and example data files, and creating the executable image. All the HILDA logicals and synonyms were set when the above mentioned login.com file was executed. In principal this login.com file should be executed each time you logon, before you run HILDA.

- Install HILDA by typing
getHILDA

In principal you need to do this only once. However, it will correctly reinstall HILDA if done more than once.

At this point all HILDA logical names and all synonyms have been defined. The HILDA directories have been defined and the HILDA files put into them. For this example the directories are:

<u>Directory</u>	<u>Contents</u>
[USER.HILDA.ALL]	source modules, if not deleted during installation
[USER.HILDA.UTILITY]	utility modules, if not deleted during installation
[USER.HILDA.EQU]	version files
[USER.HILDA.EXE]	execution file, HILDA.exe and all I/O files
[USER.HILDA.DAT]	data files and example directories
[USER.HILDA.DAT.DAT3MV]	data files for the example 3MV
[USER.HILDA.DAT.DAT3000MV]	data files for the example 3000MV
[USER.HILDA.DAT.DAT4MJ]	data files for the example 4MJ driver
[USER.HILDA.DAT.UTILITY]	data files for the HILDA utilities

You can use the HILDA synonyms to make any of these directories the default directory. This is done as shown below.

- To set the current default directory to [USER.HILDA.ALL] type
ALL
- To set the current default directory to [USER.HILDA.UTILITY] type
UTILITY

Figure 5-3. Hilda Installation (continued)

- To set the current default directory to be [USER.HILDA.DAT.UTILITY] type
DATUTILITY
- To set the current default directory to be [USER.HILDA.EQU] type
EQU
- To set the current default directory to [USER.HILDA.EXE] type
EXE

The executable image of HILDA along with the version files for the HILDA modules and the data files for the HILDA utilities are now in the directory [USER.HILDA.EXE]. Again, note that *USER.HILDA* will be whatever you have chosen above for the HILDA root directory.

To run HILDA it is necessary to have data sets for the modules. These data sets depend on the problem being run. We have furnished here three data sets. These example data sets are in [USER.HILDA.DAT].

- To set the current default directory to [USER.HILDA.DAT.DR4MJ3] type
DAT3MV
- To set the current default directory to [USER.HILDA.DAT.DR4MJ3000] type
DAT3000MV
- To set the current default directory to [USER.HILDA.DAT.DR4MJ] type
DAT4MJ

At this point HILDA is completely setup to run, except for the module data sets. These must be placed into the execution directory [USER.HILDA.EXE]. How to do this is explained in *Running HILDA*.

In principal it is the HILDA programmer who modifies any of the HILDA source code. If the user running HILDA has no need for this source code, it can be deleted. It is recommended that this be done. Updating HILDA should be done carefully, it is not to be done casually or problems will arise. If these source files were left during the installation, they can easily be deleted.

- To delete the HILDA module source files and the HILDA utility source files type
noHILDAsource

This does not delete the directory names; thus the HILDA directories still exist. The source code can be recovered by reinstalling HILDA.

For the Experienced/Expert VAX User

One way to initially setup HILDA is to logon to a computer system that has access to the LBL VAX CSA cluster and copy from the LBL CSA VAX the complete contents of the directory that contains HILDA. The present version is in

WEEK:[BENDING.HILDA]

To find the location of this directory type the command

DISKSPACE BENDING

The disk name that is on the weekly usage line is the correct name to use in place of WEEK. If necessary, a logical name can be defined using the full pathname to the LBL CSA VAX cluster, with the aforementioned disk as the device name.

Figure 5-3. Hilda Installation (continued)

This disk tool is available if you are running in the LBL VAX cluster. If you do not have access you will have to in some way determine the logical name of the disk that currently contains the HILDA files.

The complete contents of the HILDA directory can be seen by typing

DIRECTORY WEEK:[BENDING.HILDA...]

where WEEK is replaced by what is shown by DISKSPACE. Note that the device on which the files in this directory reside can change. So the above DISKSPACE inquiry is necessary to determine where HILDA is currently located.

If only the executable image of HILDA is desired, it can be copied from the directory WEEK:[BENDING.HILDA.EXE]. This directory also contains the version files for the modules and also all utility data files. It does not contain the necessary module data files. The example data files for these modules are in the directory WEEK:[BENDING.HILDA.DAT].

If no executable code is desired, then use the directory guide shown above to locate the source code and example data files. Replace USER by BENDING and COPY the desired files to you local directory; be sure to include in WEEK the pathname to the LBL disk that contains HILDA.

How to Initially Create HILDA Using the Folder TXT/Hilda/VAX

The folder Hilda/TXT contains the ASCII flat files for HILDA. Basically it is a text file image of the folder Hilda/MSW. The folders that are contained in Hilda/TXT folder have a direct correspondence to the directories on the VAX. In what follows we assume that the HILDA root directory is [USER.HILDA].

This folder-directory correspondence is show below.

<u>Folder</u>	<u>Directory</u>	<u>Contents</u>
TXT/Hilda/ALL	[USER.HILDA.ALL]	HILDA source modules
TXT/Hilda/U	[USER.HILDA.UTILITY]	HILDA utility modules
TXT/Hilda/EQU	[USER.HILDA.EQU]	HILDA version files
TXT/Hilda/DAT	[USER.HILDA.DAT]	HILDA data files
Utility Data	[USER.HILDA.DAT.UTILITY]	Utility module data files
dr4MJ@3MV	[USER.HILDA.DAT.DAT3MV]	3MV example data files
dr4MJ@3000MV	[USER.HILDA.DAT.DAT3000MV]	3000MV example data files
dr4MJData	[USER.HILDA.DAT.DAT4MJ]	4MJ driver data files
TXT/Hilda/VAX	[USER.HILDA]	HILDA VAX command files

HILDA is initially installed by downloading the ASCII text files in the folder Hilda/TXT to the computer environment in which HILDA runs. The following describes how to initially setup HILDA on the VAX.. HILDA can be installed into any FORTRAN computer environment. This VAX example can be used as a guide as to how to set up the necessary files. Also, this is an example on one way to set up the files. Obviously there are other ways to do this, the experienced computer user can install HILDA and the associated files in whatever way is convenient.

Figure 5-3. Hilda Installation (continued)

We give below step-by-step instructions for initially creating HILDA from the folder Hilda/TXT. In the following it is assumed that a file transfer program is used to transfer the complete contents of each folder to the current directory. When we say DOWNLOAD FOLDER this means to download the contents of the named folder into the current directory. We also assume that the contents of the folder HILDA/Log that is in the data folders will not be downloaded. This is also true of any ReadME files that may be in the folders. The ReadME files are not text files, they are in fact specially formatted word processor files. If they are downloaded, as may happen when using Telnet FTP, they should be deleted; their VAX image is of not useable. Remember, in each command that contains a directory reference the example will use

[USER.HILDA]

In actual practice this will be what you have chosen for the HILDA root directory. The *login.com* file will correctly set the VAX logicals and synonyms that are used below.

- Create a directory that will be the root directory for HILDA. In this example would issue the command

CREATE/DIRECTORY [USER.HILDA]

- Make this directory your default directory. In this example we would issue the command

SET DEFAULT [USER.HILDA]

- Put in this directory all the .com files that are in the folder TXT/Hilda/VAX.

DOWNLOAD TXT/Hilda/VAX

- Edit the just created LOGIN.COM file to:

- replace *rootName* in the statement

\$ HILDAroot := 'rootName

with the correct HILDA root directory name. In this example the statement becomes

\$ HILDAroot := USER.HILDA

- Save this edited file, it is the correct login.com file for HILDA.

Instead of editing the LOGIN.COM file you could type

rootName := USER.HILDA

However, this symbol definition, which is needed, will vanish when you logoff and you would have to reestablish this symbol definition each time you login to run HILDA.

- Execute this VAX command file by typing

@LOGIN.COM

If you have any error messages issued during the execution of the command file, you should check that the above quantities have been correctly entered.

- To verify that all is right before installing HILDA type

SHOW SYMBOL HILDAroot

The results returned should agree with what you have been instructed to type in the above instructions. Up to this point about the only mistake that you can make it to have incorrectly entered the requested information.

Figure 5-3. Hilda Installation (continued)

You are now ready to actually install HILDA by creating the directories, transferring the source files and example data files, and creating the executable image. All the HILDA logicals and synonyms were set when the above mentioned login.com file was executed. In principal this login.com file should be executed each time you logon, before you run HILDA.

At this point all needed logical names and all synonyms have been defined. The HILDA VAX directories are assumed to be defined. If they are not already created, they can be created by executing the command

`makeHILDAdir`

The next step is to download into the appropriate directories the files in the folders

`TXT/Hilda/ALL`

`TXT/Hilda/U`

`TXT/Hilda/EQU'`

`TXT/Hilda/DAT`

These folders are in the folder HILDA/TXT.

- Set the default directory to [USER.HILDA.ALL] by typing the command
`ALL`
- Download the Hilda source modules into this directory.
`DOWNLOAD` folder `TXT/Hilda/ALL`
- Set the default directory to be [USER.HILDA.UTILITY] by typing the command
`UTILITY`
- Download the Hilda utility source modules into this directory.
`DOWNLOAD` folder `TXT/Hilda/U`
- Set the default directory to be [USER.HILDA.DAT.UTILITY] by typing the command
`DATUTILITY`
- Download the data files for the Hilda utilities into this directory.
`DOWNLOAD` folder `Utility Data`
- Set the default directory to be [USER.HILDA.EQU] by typing the command
`EQU`
- Download the Hilda modules version files into this directory.
`DOWNLOAD` folder `TXT/Hilda/EQU`

All necessary HILDA files are now in place. The next step is to create the files for executing HILDA.

- Create the execution file for HILDA by typing the command
`makeHILDA`

The executable image of HILDA along with the Hilda module version files and the utility data files are now in the directory [USER.HILDA.EXE]. Again, note that *user.hilda* will be whatever you have chosen above for the HILDA root directory.

Figure 5-3. Hilda Installation (continued)

To run HILDA it is necessary to have data sets for the modules. These data sets depend on the problem being run. We have furnished here three data sets. These example data sets are in the folder TXT/Hilda/DAT. To get these examples we download them as follows:

- Set the default directory to [USER.HILDA.DAT.DAT3MV] by typing the command
DAT3MV
- Put the 3MV example data files into this directory.
DOWNLOAD folder dr4MJ @ 3MV
- Set the default directory to [USER.HILDA.DAT.DAT3000MV] by typing the command
DAT3000MV
- Put the 3000MV example data files into this directory.
DOWNLOAD folder dr4MJ @ 3000MV
- Set the default directory [USER.HILDA.DAT.DAT4MJ] by typing the command
DAT4MJ
- Put the 3000MV example data files into this directory.
DOWNLOAD folder dr4MJ Data

At this point HILDA is completely setup to run, except for the module data sets. These must be placed into the execution directory [USER.HILDA.EXE]. How to do this is explained in *Running HILDA*.

In principal it is the HILDA programmer who modifies any of the HILDA source code. If the user running HILDA has no need for this source code, the modules can be deleted. It is recommended that this be done. Updating HILDA should be done carefully, it is not to be done casually or problems will arise.

- Delete the HILDA module source files and the HILDA utility source files by typing the command
noHILDAsource

This completes the task of initially creating HILDA from the folders in Hilda/TXT.

END

**DATE
FILMED**

11 / 18 / 92

