doi: 10.7690/bgzdh.2021.08.019

基于相关性矩阵合并算法的系统级测试性建模方法研究

韩　露[1]，史贤俊[1]，秦玉峰[1]，林　云[1]，李荣新[2]
（1. 海军航空大学，山东 烟台 264000；2. 中国人民解放军 91663 部队，山东 青岛 266000）

摘要：针对大型复杂系统直接构建测试性模型难度较大的问题，提出一种相关性矩阵合并算法。分析大型复杂系统的层次模块划分原则，对层次模块化测试性模型的建模要素进行了简要阐述。合并算法通过对系统进行层次模块划分，然后对低层次的单元模块分别进行测试性建模，再将低层次单元模块的故障-测试相关性矩阵逐步合并，进而生成整个系统的故障-测试相关性矩阵，并以某型导弹的导航系统为例进行实例验证，简化系统级测试性模型构建的难度。

关键词：测试性；大型复杂系统；系统级；建模
中图分类号：TJ06　　**文献标志码：**A

Research on System-level Testability Modeling Method Based on Correlation Matrix Merging Algorithm

Han Lu[1], Shi Xianjun[1], Qin Yufeng[1], Lin Yun[1], Li Rongxin[2]
(1. *Navy Aviation University*, *Yantai* 264000, *China*; 2. *No*. 91663 *Unit of PLA*, *Qingdao* 266000, *China*)

Abstract: Aiming at the difficulty of building testability model directly for large complex system, a correlation matrix merging algorithm is proposed. Firstly, the principle of hierarchical module division of large complex system is analyzed, and the modeling elements of hierarchical modular testability model are briefly described. The merging algorithm divides the hierarchical modules of the system, and then builds testability models for the low-level unit modules. The fault-test correlation matrix of the low-level unit module is gradually merged to generate the fault test correlation matrix of the whole system. Finally, the navigation system of a certain missile is taken as an example to verify, which simplifies the difficulty of system level testability model construction.

Keywords: testability; large complex system; system level; modeling

0　引言

目前，大型复杂系统普遍具有以下设计特点：1) 层次化的设计结构。大型复杂系统的设计具有一定的层次结构，高层次的单元或模块由各个低层次的子模块共同组成[1-2]。2) 模块化的设计结构。一个模块的主要功能由其子功能模块集成共同完成，如导弹红外成像导引装置由随动稳定成像装置、控制跟踪装置、电缆、二次电源等子模块组成，其功能也由这些子模块共同完成[3]。目前大多学者都是以最低层次的组成单元为对象，建立整个系统的故障-测试相关性矩阵[4]。实际上，大型系统的复杂性决定了其建模过程中难以直接确定故障与信号的相关关系[5]；因此，笔者在理想条件的假设下，提出了一种相关性矩阵合并算法，通过对不同层次的功能模块分别进行测试性建模，然后由低层次故障-测试相关性矩阵合并生成整个系统的故障-测试相关性矩阵。该方法能够有效地降低计算大型复杂系统相关性矩阵的复杂程度。

1　复杂系统结构划分方法

复杂系统一般具有层次化和模块化的性质。系统划分的基本原则是以测试维修保障体制和系统的组成结构为基础，达到简化维修测试活动的目的。对系统进行层次模块划分是为了更好地对装备进行测试维修活动[6]。而文中的模块划分，从故障诊断和维修保障的角度来看，实际上是系统功能模块的二次组合，目的是便于进行故障测试诊断和维修，不完全遵循系统的物理结构。

1.1　层次划分

测试性设计目的是使系统具有良好的测试性，更好地进行测试维修活动。由于已经对系统的测试维修活动级别进行了要求，所以在系统测试性设计阶段也提出了对系统进行功能层次划分的需求，保证划分的各层次与测试维修需求相匹配。这样保证

收稿日期：2021-04-30；**修回日期：**2021-05-28
作者简介：韩　露(1995—)，男，山东人，硕士，从事装备测试性设计、故障诊断、故障检测研究。E-mail: onewest@163.com。

了系统能够满足不同场合的测试维修需求[7]。

在对系统进行测试性建模之前，约定层次划分是否合理影响到后续武器装备的测试性工作能否顺利开展。为了使武器装备能够进行合理的约定层次划分，保证武器装备可以顺利进行各级别的测试维修活动，应按照以下要求划分系统的约定层次：

1) 为保证符合系统约定层次划分与测试维修级别相对应，应采用自上向下的划分原则，可以保证层次间的故障诊断能够符合测试诊断逻辑。若系统发生了一种故障，该故障所在层次就会受该故障影响而体现出相应的故障特征，例如在较低层次的故障特征体现为元件故障，较高层次则体现为某些组件出现故障等。低层次故障与高层次故障相比，由于与具体部件相关联，因此更加详细。系统故障的诊断隔离顺序为先在较高层次进行故障隔离定位，再对较低层次单元进行故障隔离定位。这种将故障模式由高到低的定位方法能够描述系统各层次之间的关联关系，可以清晰地描述系统故障传播空间中各层次故障模式之间的因果关系。

2) 必须根据系统内功能信息的相关关系进行层次划分。按照由上至下、由大到小的思想，考虑系统自身的结构复杂程度和测试性设计要求，系统的总体功能由若干子系统功能相互叠加共同实现，而每个子系统功能又可以向更低的层次划分，最终划分至最低层次所对应的最简单功能[8]。各单元在同一层次内相互独立，在高低层次之间则是包含关系：一个低层次单元应是其上一层次单元的组成部分，一个高层次单元包含了若干低层次功能单元。另外，必须要根据系统结构组成特点对系统各层单元的功能进行定义和划分，保证测试性设计分析工作能够顺利进行。

3) 必须根据测试维修、故障隔离的要求进行层次划分。武器装备在不同维修级别所要求达到的故障诊断精度也不相同，根据分层诊断、分级维修的原则，不同的测试维修、故障隔离的要求决定了武器装备的分解层次，即只需将系统分解至相应级别所要求隔离到的最小可更换单元所在的层次即可。在基层级维修时，由于测试设备水平较低且使用环境条件受限，能够将故障隔离定位到 LRU 层即可，因此只需要相对应地将层次划分到 LRU 层；中继级和基地级则分别需要将层次划分到 SRU 层和元件层。系统的测试性设计也需要以系统的实际功能结构为基础，将系统划分为 LRU、SRU 和元件 3 个层次并分别进行测试性分析，确保能够达到各维修级别要求的维修测试精度，缩短测试诊断时间，提高测试维修活动的效率。

4) 分别定义各层次的故障模式。由于故障传播方向仅能由低层次故障向高层次传播，因此高层次单元发生故障的原因必然是因为低层次单元发生了故障，即最低层次单元发生故障是其他层次单元发生故障的根本原因[9-10]。对同一个故障来说，在不同的层次模块中会表现出不同的故障状态，由于各个维修级别所关心的故障表现形式和能够更换的备件也都不同，因此可以把低层次的故障模式所引起较高层次的故障定义为较高层次的故障模式，即分别定义不同层级的故障模式，故障模式可以在不同层次之间由低层次向高层次传播但又互相独立。在进行测试性设计分析时，仅需要在相应层次进行设计分析，这样既能够符合测试维修的级别划分，又可以降低测试维修规模，提高工作效率。

按照上述思想和标准对各个层次分别进行测试性建模和测试性设计，使各层次都能达到预期的测试性指标，这样不但使系统功能层次结构的描述更加明确，同时还可以结合各层次单元的功能和现场测试维修环境，保证了各测试维修级别的工作能够顺利进行，达到提高工作效率、减少测试维修时间、降低测试诊断复杂程度、保证系统的可靠性、测试性等达到要求，降低全寿命周期费用的目的。

多信号流图模型的建模过程采用了分层建模的思想，根据系统实际结构和需求可将系统划分为子系统层、LRU 层、SRU 层、元件层和故障模式层等。以图 1 所示为例，某导弹红外导引头上红外成像导引装置内俯仰机构的某个陀螺存在零漂过大这一故障，该元件层故障可以导致 SRU 层单元随动稳定装置自检异常，进而导致 LRU 层红外成像导引装置自检、射检异常，最后导致整个系统无法正常工作。

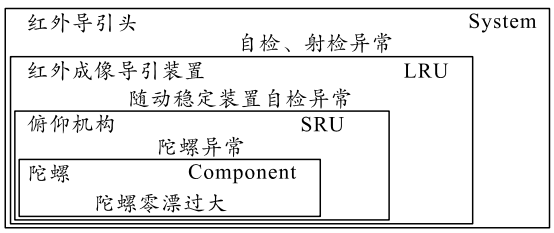


图 1　红外导引头层次划分部分

1.2　模块划分

由于系统本身设计等因素，有时存在内部模块耦合严重等问题，使得难以对故障进行隔离定位；因此，在已知系统约定层次的情况下，还应以系统功能组成结构为基础，以简化测试和故障诊断活动

为原则，在各层次内进行功能子模块的划分。下面给出一个简单的子模块划分的示例。以某系统为例，该系统的部分结构模型如图 2 所示。

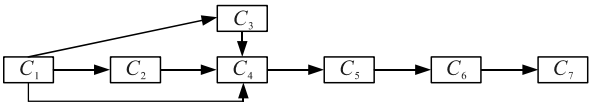
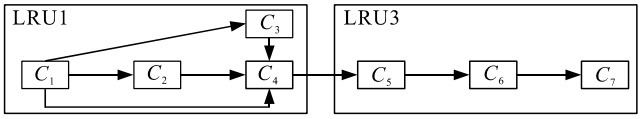
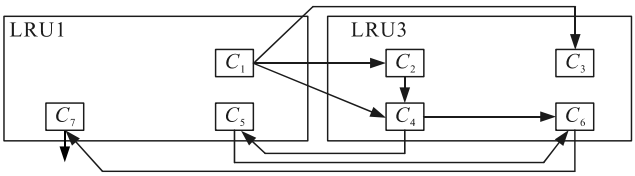


图 2 系统结构模型

对比图 3(a)与 3(b)，显然图 3(a)中模块的划分更加合理，2 个子模块(LRU1、LRU3)之间直接相连，可以很容易地对 2 个子模块的故障进行隔离；而图 3(b)中 2 个子模块存在反馈连接关系，从而难以将故障隔离到这 2 个子模块，需要重新进行模块划分。



(a) 系统子模块划分方法(第 1 种)



(b) 系统子模块划分方法(第 2 种)

图 3 系统子模块划分

2 复杂系统的测试性建模方法

2.1 层次模块化测试性建模要素分析

层次模块化测试性模型的建模要素，需要对多信号流图模型的建模要素进行层次模块化的属性扩展。假设系统共有 L 个层级，则令 $l\in[1,L]$，l 代表处于系统的第 l 层级。

定义 1 层次模块集。表示形式为：

$$M=\left\{M_1^i,M_2^i,\cdots,M_l^i,\cdots,M_L^i\right\}。\tag{1}$$

式中：M 表示系统所有层次的模块集合；M_l^i 表示第 l 层的模块集，其中：i=1,2,⋯,m，代表第 l 层具有 i 个子模块；M_l^i 既可以是单个故障模块，也可是模块划分后所的组成单元。

定义 2 层次故障集。表示形式为：

$$F=\left\{F_1^i,F_2^i,\cdots,F_l^i,\cdots,F_L^i\right\}。\tag{2}$$

式中：F 表示系统所有层次的故障模式集合；F_l^i 表示第 l 层的故障模式集，其中：i=1,2,⋯,n，代表第 l 层具有 i 个故障模式。

定义 3 层次测点集。表示形式为：

$$TP=\left\{TP_1^i,TP_2^i,\cdots,TP_l^i,\cdots,TP_L^i\right\}。\tag{3}$$

式中：TP 表示系统所有层次的测试点集合；TP_l^i 表示第 l 层的测试点集，其中：i=1,2,⋯,r，代表第 l 层有 i 个测试点。

定义 4 层次测试集。表示形式为：

$$T=\left\{T_1^i,T_2^i,\cdots,T_l^i,\cdots,T_L^i\right\}。\tag{4}$$

式中：T 表示系统所有层次的测试集合；T_l^i 是第 l 层的测试集，其中：i=1,2,⋯,s，表示第 l 层有 i 个测试。

依据上述层次化测试性模型的建模要素分析，系统第 l 层的测试性模型可用式(3)-(5)来表示：

定义 5 层次测试性模型。表示形式为：

$$G=(M_l,F_l,TP_l,T_l)。\tag{5}$$

层次模块化测试性模型的建模过程为：1) 分析高层次单元模块的建模要素，建立系统高层次的测试性模型；2) 对高层次的组成模块分别建模，完成中间层次的测试性建模；3) 根据层次关系及实际需要，分析至更低层次。其中：对建立任一层次的测试性模型时，应当仅分析该层次单元模块的故障模式，而不考虑相对较低层次的故障模式。层次模块化测试性模型建模流程如图 4 所示。

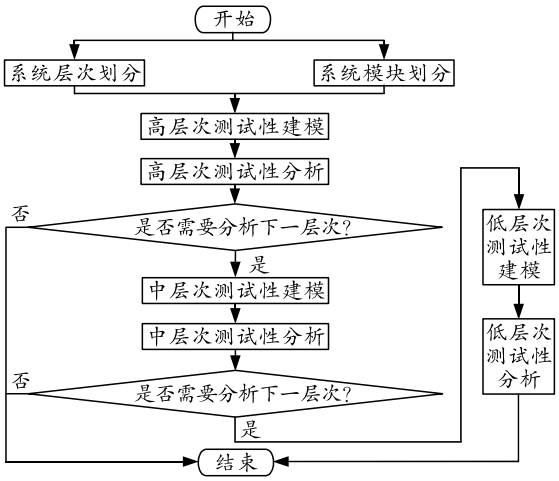
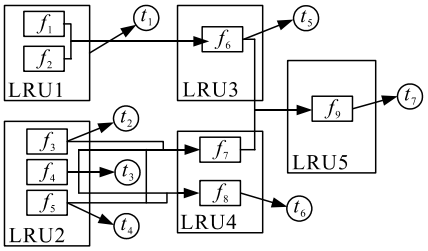
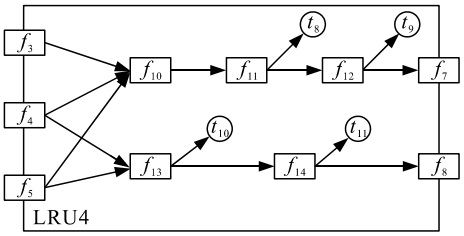


图 4 层次化测试性建模流程

仍以 1.2 节中某系统为例，说明层次模块化测试性模型建模过程。系统的结构图示模型如图 5。



(a) LRU 层结构模型



(b) LRU4 结构模型

图 5　结构模型

该系统有 5 个 LRU 层模块，分别为 LRU1～LRU5。由图 5(a)可知，5 个 LRU 层模块共有 9 个故障模式，分别为 f_1～f_9；共有 7 个 LRU 层测试，分别为 t_1～t_7。其中：LRU4 内部结构如图 5(b)所示。由图 5(b)可知，共有 5 个 SRU 层故障模式，分别为 f_{10}～f_{14}；共有 4 个 SRU 层测试，分别为 t_8～t_{11}。以 LRU 层为指定分析对象，得到 LRU 层故障-测试相关性矩阵如下所示：

$$\begin{array}{c} \\ f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \\ f_7 \\ f_8 \\ f_9 \end{array} \begin{array}{c} \begin{array}{ccccccc} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array} 。 \tag{6}$$

然后指定分析对象为 LRU4，分析层次为 SRU 层，LRU4 的故障-测试相关性矩阵如下所示：

$$\begin{array}{c} \\ f_{10} \\ f_{11} \\ f_{12} \\ f_{13} \\ f_{14} \end{array} \begin{array}{c} \begin{array}{cccc} t_8 & t_9 & t_{10} & t_{11} \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{array} 。 \tag{7}$$

可以看出，层次模块化测试性建模直观地表达了系统各层次的故障和测试的相关关系，能够满足不同级别的测试维修需要。例如对大型复杂系统进行高层次测试性分析，仅对相应层次的故障测试相关关系进行分析即可，而无需考虑较低层次的故障模式和测试；因此，适用于基层级的测试维修活动，说明了层次模块化测试性建模方法的实用性。

2.2　相关性矩阵合并算法

上节中介绍了系统测试性建模的一般方法，然而对大型复杂系统来说，其组成元件数量众多，模块与信号的相关关系也极为复杂；因此，无论是基于专家经验，还是基于物理模型做定量仿真，如果直接去获取整个系统的故障-测试相关性矩阵，工作难度极高。考虑到多信号流图模型的分层建模思想，笔者提出一种故障-测试相关性矩阵的合并算法：先对较低层次的子模块建立低层次的多信号流图模型，并获得低层次模块的故障-测试相关性矩阵，再将各子模块的矩阵进行合并计算，进而生成整个系统的故障-测试相关性矩阵。

对于相对较低层次的各个子模块，其故障-测试相关性矩阵可以通过简单分析获得。之后，根据划分模块的顺序，进行对角化连接，构成一个模块对角矩阵 $\boldsymbol{D}^*$：

$$\boldsymbol{D}^*=\begin{bmatrix} \boldsymbol{D}_1 & \boldsymbol{Y} \\ \boldsymbol{X} & \boldsymbol{D}_2 \end{bmatrix}。 \tag{8}$$

式中：$\boldsymbol{D}_1$ 为第一个子模块的故障-测试相关性矩阵；$\boldsymbol{D}_2$ 为第一个子模块后续连接子模块的故障-测试相关性矩阵。

若子模块划分时，模块与模块之间不存在反馈连接，即上一子模块中的测试无法检测后续子模块中发生的故障，则此时分块矩阵 $\boldsymbol{X}$=0；因此只需对矩阵 $\boldsymbol{Y}$ 进行计算即可确定矩阵 $\boldsymbol{D}^*$。矩阵 $\boldsymbol{Y}$ 内的元素 y_{ij} 可通过如下原则进行确定：

若前一子模块中某节点故障是全局故障，此时：

$$y_{ij}=ft_{ij}。 \tag{9}$$

若前一子模块中某节点故障是功能故障，此时：

$$y_{ij}=ft_{ij}\times fst_{ij}。 \tag{10}$$

式中：i 为系统中第 i 个故障模式；j 为系统中第 j 个测试。ft_{ij}=1 表示第 i 个故障模式至少存在一条有向通路能够到达第 j 个测试，ft_{ij}=0 表示第 i 个故障与第 j 个测试之间没有任何有向通路。若第 i 个故障模式能够影响的信号集 SM_i 与第 j 个测试能够检测的信号集 ST_j 有交集，即 $SM_i \cap ST_j \neq \varnothing$，则 fst_{ij}=1；若信号集 SM_i 与信号集 ST_j 没有公共信号，即 $SM_i \cap ST_j=\varnothing$，则 fst_{ij}=0。

若子模块划分时，模块与模块之间存在反馈连接，需对矩阵 $\boldsymbol{X}$、$\boldsymbol{Y}$ 分别进行计算才能确定矩阵 $\boldsymbol{D}^*$。矩阵 $\boldsymbol{Y}$ 内元素的计算方法同上，矩阵 $\boldsymbol{X}$ 内的元素 x_{ij} 可通过如下原则进行确定：

若后一子模块中某节点故障是全局故障，此时：

$$x_{ij}=ft_{ij}。 \tag{11}$$

若后一子模块中某节点故障是功能故障，此时：

$$x_{ij} = ft_{ij} \times fst_{ij}。 \tag{12}$$

由式(8)—(10)可计算得到模块对角矩阵 $\boldsymbol{D}^*$，若此时存在后续第三个子模块的故障-测试相关性矩阵 $\boldsymbol{D}_3$，则可在 $\boldsymbol{D}^*$的基础上，再次构建模块对角矩阵：

$$\boldsymbol{D}^{**}=\begin{bmatrix}\boldsymbol{D}^* & \boldsymbol{Y}^*\\ \boldsymbol{X}^* & \boldsymbol{D}_3\end{bmatrix}。 \tag{13}$$

以此类推，最终生成系统级故障-测试相关性矩阵。算法实现流程如图 6 所示。

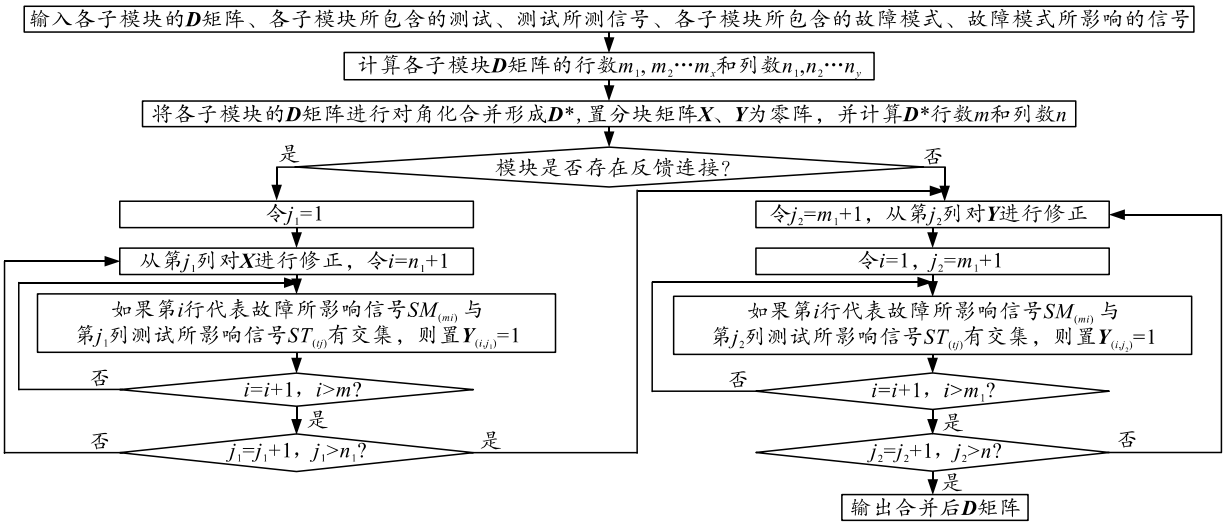


图 6 故障-测试相关性矩阵合并算法流程

3 实例验证

某惯导系统具有明显的层次化和模块化结构，由于其内部组成元件的故障模式较多，组成元件连接关系复杂，直接获取整个系统的故障-测试相关性矩阵比较困难；因此，可以采用层次模块化测试性模型的建模方法对整个系统的故障-测试相关性矩阵进行计算。在功能上，系统主要由加速度计和陀螺仪测量 3 个坐标轴上的线速度和角速度，经误差补偿后，将线速度、角速度及增量输出至中心信息处理器，然后计算导弹的运动参数，调节控制导弹的运动姿态。在物理组成上，系统由加速度计、陀螺仪、A/D 转换电路、滤波电路、二次电源等 5 部分组成。其组成框图如图 7 所示。

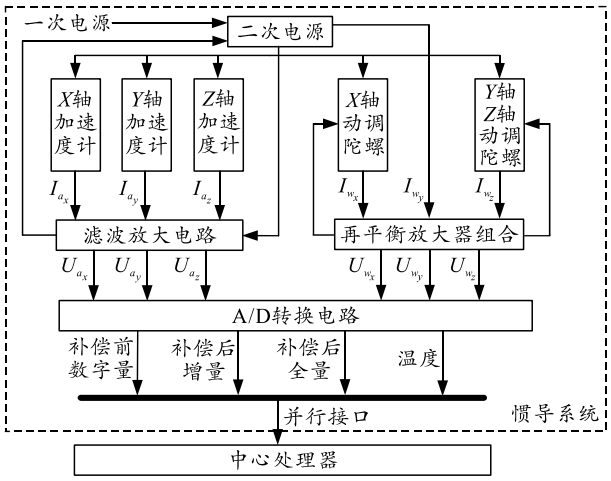


图 7 惯导系统组成

系统共有 29 个故障模式及 27 个备选测试，分别用 f_1～f_{29}、t_1～t_{27} 表示，如表 1、表 2 所示。

表 1 惯导系统故障模式

模块名称	故障模式	故障模式编号
X 轴加速度计	X 轴加速度计无输出或满量程	f_1
	X 轴加速度计零位漂移	f_2
Y 轴加速度计	Y 轴加速度计无输出或满量程	f_3
	Y 轴加速度计零位漂移	f_4
Z 轴加速度计	Z 轴加速度计无输出或满量程	f_5
	Z 轴加速度计零位漂移	f_6
YZ 轴陀螺再平衡放大器组合	Y 轴信号放大器无输出或满量程	f_7
	Z 轴信号放大器无输出或满量程	f_8
A/D 转换电路	并行接口故障	f_{29}
二次电源	三相方波无输出	f_9
	+35 V 电源无输出	f_{10}
	−35 V 电源无输出	f_{11}
	5VAG 偏置电压无输出	f_{12}
	5VDG 偏置电压无输出	f_{13}
	+15 V 电源无输出	f_{14}
	−15 V 电源无输出	f_{15}
	7 V 16 kHz 信号源无输出	f_{16}
	6 kHz 频标无输出	f_{17}
X 轴动调陀螺	陀螺 1 的 X 轴无输出或满量程	f_{18}
	陀螺 1 的 X 轴零位漂移	f_{19}
	陀螺 2 的 Y 轴无输出或满量程	f_{20}
YZ 轴动调陀螺	陀螺 2 的 Z 轴无输出或满量程	f_{21}
	陀螺 2 的 Y 轴零位漂移	f_{22}
	陀螺 2 的 Z 轴零位漂移	f_{23}
X 轴陀螺再平衡放大器组合	X 轴信号放大器无输出或满量程	f_{24}
滤波放大电路	+27 V 无输出	f_{25}
	X 轴无输出或满量程输出	f_{26}
	Y 轴无输出或满量程输出	f_{27}
	Z 轴无输出或满量程输出	f_{28}

表 2　惯导系统备选测试表

序号	备选测试
t_1	Z 轴加速度计输出信号测试
t_2	Y 轴加速度计输出信号测试
t_3	X 轴加速度计输出信号测试
t_4	Y 轴陀螺输出信号测试
t_5	X 轴陀螺输出信号测试
t_6	X 轴再平衡放大输出测试
t_7	Z 轴陀螺输出信号测试
t_8	Z 轴滤放大输出信号测试
t_9	Y 轴滤放大输出信号测试
t_{10}	X 轴滤放大输出信号测试
t_{11}	Y 轴再平衡放大输出测试
t_{12}	Z 轴再平衡放大输出测试
t_{13}	并口信号输出信号测试
t_{14}	27 V 滤波放大输出信号测试
t_{15}	二次电源中三相方波测试
t_{16}	二次电源中+35 V 电源测试
t_{17}	二次电源中 7 V 16 kHz 电源测试
t_{18}	二次电源中+5VAG 电源测试
t_{19}	二次电源中+5VDG 测试
t_{20}	二次电源中+15 V 电源测试
t_{21}	二次电源中 16 kHz 频标测试
t_{22}	X 轴陀螺零位漂移测试
t_{23}	Y 轴陀螺零位漂移测试

为便于分析，将系统划分为系统级(惯导系统)、LRU 级(加速度计、陀螺、再平衡放大组合器、滤波放大电路、A/D 转换电路、二次电源)、故障模式级 3 个层次；将各 LRU 按组成结构划分为 M_1、M_2、M_3、M_4 4 个模块，模块划分如图 8 所示。

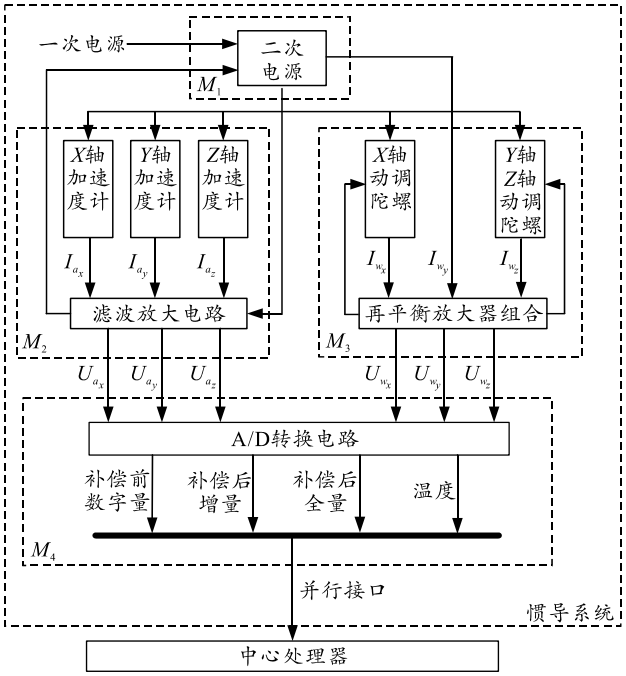


图 8　惯导系统模块划分

分别建立各模块的测试性模型，并依次得到模块 M_1、M_2、M_3、M_4 的故障-测试相关性矩阵，如下所示：

$$\boldsymbol{D}_1=\begin{array}{c} \\ f_9\\ f_{10}\\ f_{11}\\ f_{12}\\ f_{13}\\ f_{14}\\ f_{15}\\ f_{16}\\ f_{17}\end{array}\begin{array}{c}\begin{array}{ccccccc} t_{15} & t_{16} & t_{17} & t_{18} & t_{19} & t_{20} & t_{21}\end{array}\\ \begin{bmatrix}1&0&0&0&0&0&0\\0&1&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&0&1&0&0&0\\0&0&0&0&1&0&0\\0&0&0&0&0&1&0\\0&0&0&0&0&0&0\\1&0&1&0&0&0&0\\0&0&0&0&0&0&1\end{bmatrix}\end{array};$$

$$\boldsymbol{D}_2=\begin{array}{c} \\ f_1\\ f_2\\ f_3\\ f_4\\ f_5\\ f_6\\ f_{25}\\ f_{26}\\ f_{27}\\ f_{28}\end{array}\begin{array}{c}\begin{array}{cccccccccc} t_1 & t_2 & t_3 & t_{25} & t_{26} & t_{27} & t_8 & t_9 & t_{10} & t_{14}\end{array}\\ \begin{bmatrix}0&0&1&0&0&0&0&0&1&0\\0&0&0&1&0&0&0&0&1&0\\0&1&0&0&0&0&0&1&0&0\\0&0&0&0&1&0&0&1&0&0\\1&0&0&0&0&0&1&0&0&0\\0&0&0&0&0&1&1&0&0&0\\0&0&0&0&0&0&0&0&0&1\\0&0&0&0&0&0&0&0&1&0\\0&0&0&0&0&0&0&1&0&0\\0&0&0&0&0&0&1&0&0&0\end{bmatrix}\end{array};$$

$$\boldsymbol{D}_3=\begin{array}{c} \\ f_7\\ f_8\\ f_{18}\\ f_{19}\\ f_{20}\\ f_{21}\\ f_{22}\\ f_{23}\\ f_{24}\end{array}\begin{array}{c}\begin{array}{ccccccccc} t_4 & t_5 & t_6 & t_7 & t_{11} & t_{12} & t_{22} & t_{23} & t_{24}\end{array}\\ \begin{bmatrix}0&0&0&0&1&0&0&0&0\\0&0&0&0&0&1&0&0&0\\0&1&1&0&0&0&0&0&0\\0&0&1&0&0&0&1&0&0\\1&0&0&0&0&1&0&0&0\\0&0&0&1&1&0&0&0&0\\0&0&0&0&0&1&0&1&0\\0&0&0&0&1&0&0&0&1\\0&0&1&0&0&0&0&0&0\end{bmatrix}\end{array};$$

$$\boldsymbol{D}_4=\begin{array}{c} \\ f_{29}\end{array}\begin{array}{c}\begin{array}{c} t_{13}\end{array}\\ \begin{bmatrix}1\end{bmatrix}\end{array}。$$

将 M_2 模块与 M_3 模块进行合并生成新的模块 M_5，计算得到 M_5 的故障-测试相关性矩阵为：

	t_1	t_2	t_3	t_{25}	t_{26}	t_{27}	t_8	t_9	t_{10}	t_{14}	t_4	t_5	t_6	t_7	t_{11}	t_{12}	t_{22}	t_{23}	t_{24}
f_1	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
f_2	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
f_3	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
f_4	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
f_5	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
f_6	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
f_{25}	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
f_{26}	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
f_{27}	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
f_{28}	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
f_7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
f_8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
f_{18}	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0
f_{19}	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0
f_{20}	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0
f_{21}	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
f_{22}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0
f_{23}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1
f_{24}	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0

。

将 M_1 模块与 M_5 模块进行合并生成新的模块 M_6，计算得到 M_6 的故障-测试相关性矩阵为：

	t_{15}	t_{16}	t_{17}	t_{18}	t_{19}	t_{20}	t_{21}	t_1	t_2	t_3	t_{25}	t_{26}	t_{27}	t_8	t_9	t_{10}	t_{14}	t_4	t_5	t_6	t_7	t_{11}	t_{12}	t_{22}	t_{23}	t_{24}
f_9	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
f_{10}	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0
f_{11}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0
f_{12}	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f_{13}	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f_{14}	0	0	0	0	0	1	0	1	1	1	1	1	1	1	1	1	0	0	0	1	0	1	1	0	0	0
f_{15}	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	1	0	1	1	0	0	0
f_{16}	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
f_{17}	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0
f_1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
f_2	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
f_3	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
f_4	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
f_5	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
f_6	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
f_{25}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
f_{26}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
f_{27}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
f_{28}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
f_7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
f_8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
f_{18}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0
f_{19}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0
f_{20}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0
f_{21}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
f_{22}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0
f_{23}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1
f_{24}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0

。

最后将模块 M_6 与 M_4 合并，合并后即为整个惯导系统的故障-测试相关性矩阵，将矩阵中的行列按顺序排列后，得到惯导系统级故障-测试相关性矩阵如下所示：

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}	t_{16}	t_{17}	t_{18}	t_{19}	t_{20}	t_{21}	t_{22}	t_{23}	t_{24}	t_{25}	t_{26}	t_{27}
f_1	0	0	1	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f_2	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0
f_3	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f_4	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0
f_5	1	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f_6	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
f_7	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f_8	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f_9	0	0	0	1	1	1	1	0	0	0	1	1	1	0	1	0	0	0	0	0	0	1	1	1	0	0	0
f_{10}	0	0	0	0	0	1	0	0	0	0	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
f_{11}	0	0	0	0	0	1	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f_{12}	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0
f_{13}	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0
f_{14}	1	1	1	0	0	1	0	1	1	1	1	1	1	0	0	0	0	0	0	1	0	0	0	0	1	1	1
f_{15}	1	1	1	0	0	1	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1	1	1
f_{16}	0	0	0	1	1	1	1	0	0	0	1	1	1	1	1	0	1	0	0	0	0	1	1	1	0	0	0
f_{17}	0	0	0	0	0	1	0	0	0	0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0
f_{18}	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f_{19}	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0
f_{20}	0	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f_{21}	0	0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f_{22}	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0
f_{23}	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0
f_{24}	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f_{25}	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
f_{26}	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f_{27}	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f_{28}	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f_{29}	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

。

根据惯导系统故障-测试相关性矩阵对系统进行相关性分析。由故障-测试相关性矩阵可知，系统无未检测故障、无冗余测试，在理想条件下，惯导系统的测试性指标 γ_{FD}=100%，γ_{FI}=100%，能够达到要求。利用文献[11]提出的 FST 方法(基于多信号流图的改进建模方法)对该系统进行建模，得到系统层级的故障-测试相关性矩阵，经对比笔者所建立的矩阵相同。表 3 记录了 2 种方法的对比分析。

表 3　建模方法对比

方法	工作量	建模时长
本文中方法	分析各模块内部的故障、测试关系	较短，本案例 6 h
FST 方法	分析整个系统的故障、测试关系	长，本案例 20 h

根据上表可以看出，2 种方法的工作量不同，FST 方法需要对整个系统具有复杂耦合关系的故障、测试进行分析。而笔者提出的基于相关性矩阵合并算法的测试性建模方法仅需将系统合理划分为 4 个模块后，依次计算出各模块的相关性矩阵，最后利用合并算法可直接得到系统级的模型，大大降低了研发人员的工作量。由于 2 种方法最终所建立的模型相同，证明文中所提的测试性模型构建方法是可行的。根据多信号流图模型分层建模的思想，将整个系统分为各个子模块，对各个子模块分别建模，再将各子模块的故障-测试相关性矩阵进行合并。该方法回避了直接确定系统内故障与信号相关关系的难点，且最后所建模型能够满足系统的测试性指标，在一定程度上降低了系统级建模的复杂程度，证明了合并算法的有效性与可用性。

（下转第 91 页）