CLAIMS

1. A test instrument comprising:

a first processing system that is programmable to run one or more test programs to test a device interfaced to a test instrument, and that is programmed to control operation of the test instrument; and

a second processing system that is dedicated to device testing, the second processing system being programmable to run one or more test programs to test the device;

wherein the first processing system has a first application programming interface (API) and the second processing system has a second API, the first API and the second API being different APIs, the first API and the second API having at least some duplicate functions.

2. The test instrument of claim 1, wherein functions of the first API are associated with corresponding functions of the second API.

3. The test instrument of claim 2, wherein functions of the first API are configured to call functions of the second API in response to information received by the first processing system.

4. The test instrument of claim 3, wherein the information comprises programming code written using functions of the first API, the functions of the first API calling functions of the second API to configure the programming code to run on the second processing system.

5. The test instrument of claim 4, wherein configuring the programming code to run on the second processing system comprises generating code using functions associated with the second API in response to use of corresponding functions of the first API.

6. The test instrument of claim 1, wherein the first processing system is programmable to generate a user interface, the user interface for interacting with the first processing system and for interacting with the second processing system through the first API and the second API.

7. The test instrument of claim 1, wherein the first processing system is programmable to generate a user interface, the user interface for interacting with the second processing system through the second API.

8. The test instrument of claim 1, wherein the first processing system is programmed to run one or more test programs to test the device interfaced to the test instrument,

wherein the second processing system is not programmed to run one or more test programs to test the device.

9. The test instrument of claim 1, wherein the first processing system is not programmed to run one or more test programs to test the device interfaced to the test instrument,

wherein the second processing system is programmed to run one or more test programs to test the device.

10. The test instrument of claim 1, further comprising:

one or more ports configured to mate to the device;

wherein the first processing system is configured to send test data to the device through the one or more ports; or

wherein the second processing system is configured to send test data to the device through the one or more ports.

11. The test instrument of claim 1, wherein the first processing system comprises one or more microprocessors programmed to run a general purpose operating system; and

wherein the second processing system comprises one or more embedded microprocessors, each of the one or more embedded microprocessors being programmed to test a corresponding device interfaced to the test instrument.

12. A method performed on a test instrument, the method comprising:

receiving a test program at a first processing system that is programmable to run one or more test programs to test a device interfaced to a test instrument, and that is programmed to control operation of the test instrument;

generating, based on the test program, an altered test program that is executable on a second processing system, the second processing system being dedicated to device testing and being programmable to run one or more test programs to test the device; and

transmitting the altered test program to the second processing system.

13. The method of claim 12, wherein the first processing system has a first application programming interface (API) and the second processing system has a second API, the first API and the second API being different APIs, the first API and the second API having at least some duplicate functions.

14. The method of claim 13, wherein the test program comprises one or more calls to functions of the first API.

15. The method of claim 14, wherein generating the altered test program comprises identifying one or more corresponding functions of the second API that correspond to the one or more functions of the first API.

16. The method of claim 15, wherein generating the altered test program comprises replacing the one or more calls to functions of the first API with one or more calls to corresponding functions of the second API.

17. The method of claim 12, further comprising executing the altered test program on the second processing system to test the device.

18. The method of claim 12, further comprising providing a development environment associated with the second processing system that allows the altered test program to be modified at the second processing system.

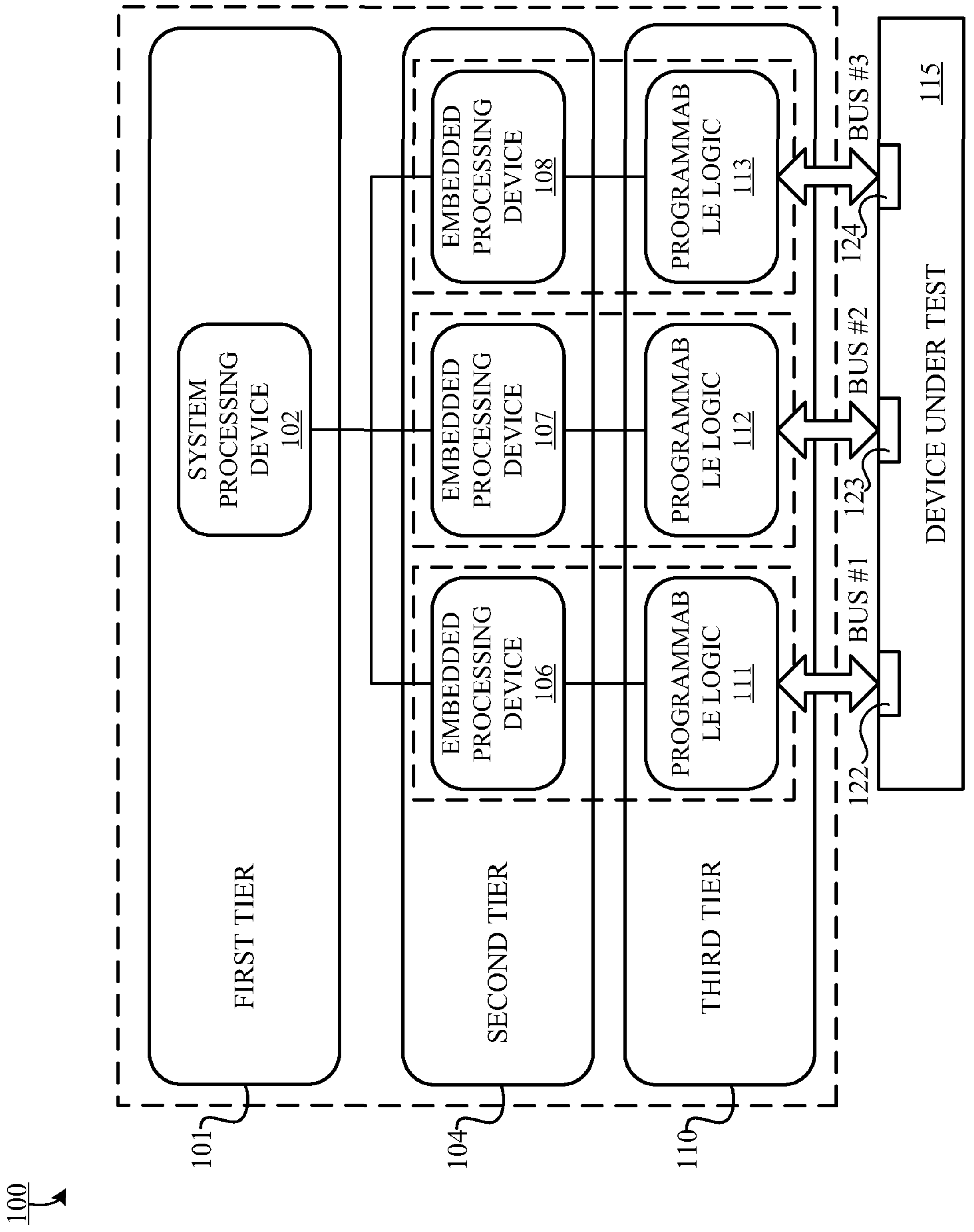
100
101
104
110
FIRST TIER
SECOND TIER
THIRD TIER
SYSTEM PROCESSING DEVICE 102
EMBEDDED PROCESSING DEVICE 106
EMBEDDED PROCESSING DEVICE 107
EMBEDDED PROCESSING DEVICE 108
PROGRAMMABLE LOGIC 111
PROGRAMMABLE LOGIC 112
PROGRAMMABLE LOGIC 113
BUS #1
BUS #2
BUS #3
122
123
124
DEVICE UNDER TEST
115

FIG. 1

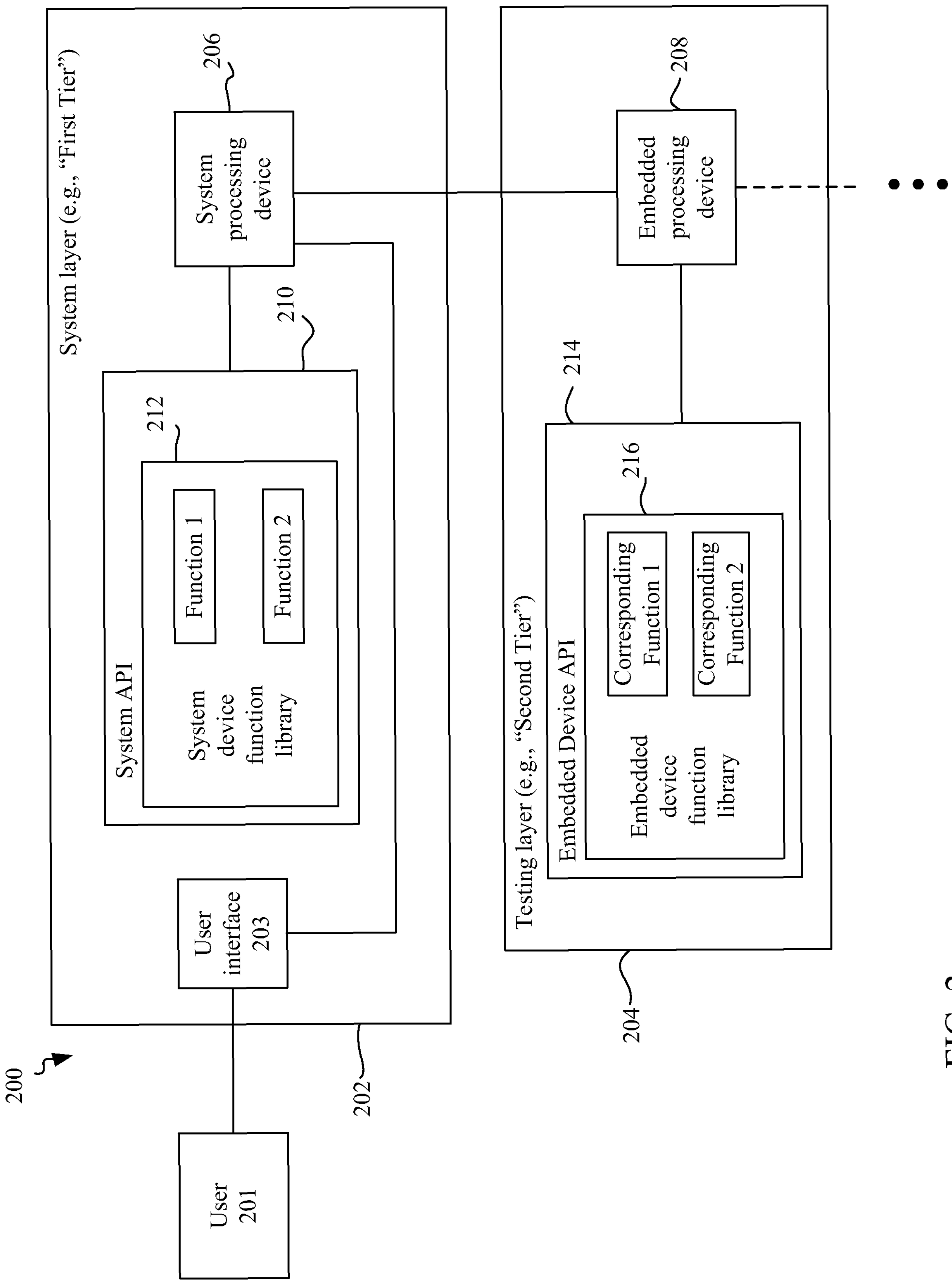
200
System layer (e.g., "First Tier")
202
System processing device
206
System API
210
System device function library
212
Function 1
Function 2
User interface 203
User 201
Testing layer (e.g., "Second Tier")
204
Embedded processing device
208
Embedded Device API
214
Embedded device function library
216
Corresponding Function 1
Corresponding Function 2

FIG. 2

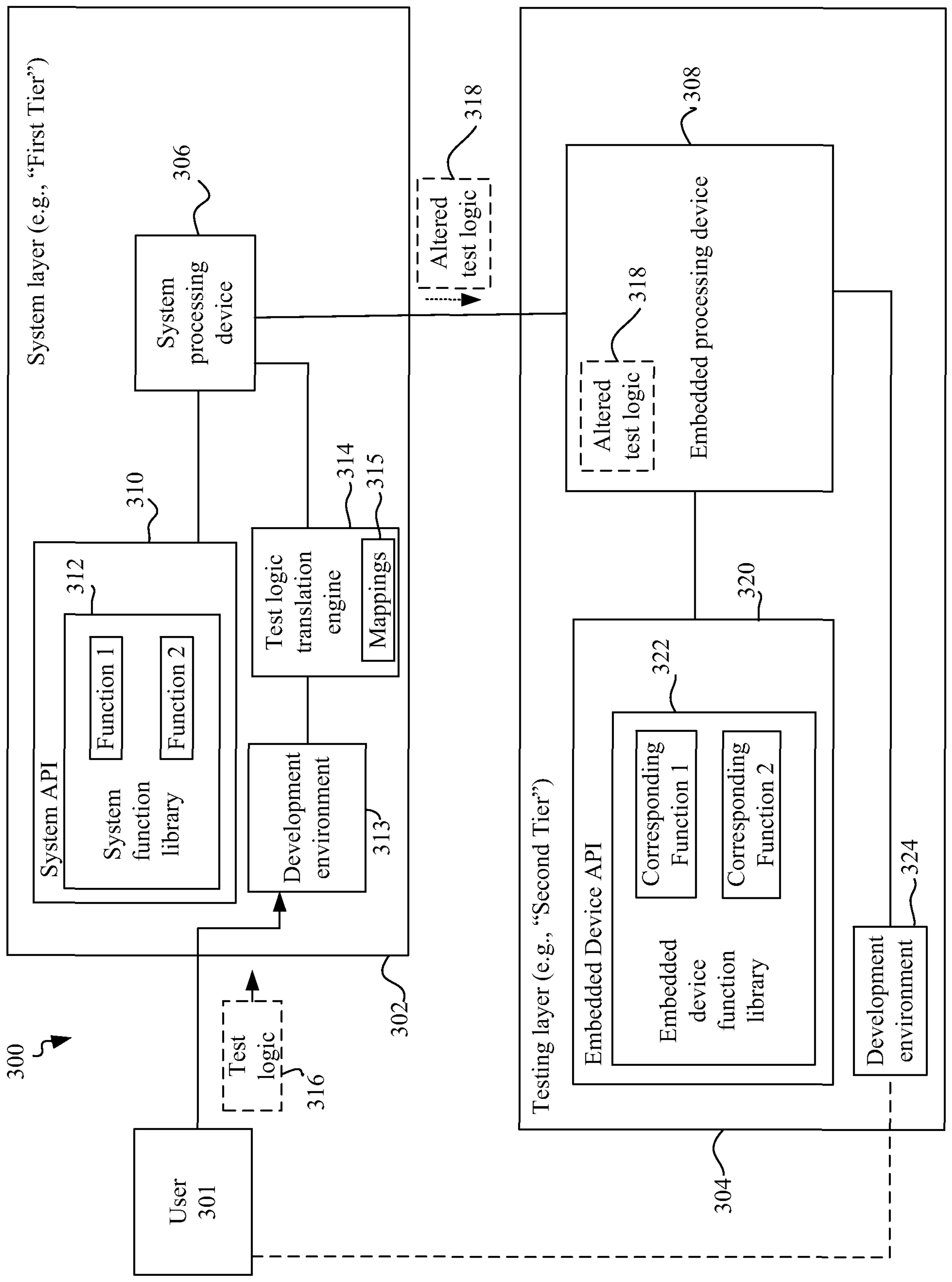
300
User 301
Test logic
316
System layer (e.g., "First Tier")
302
System processing device
306
System API
310
System function library
312
Function 1
Function 2
Development environment
313
Test logic translation engine
314
Mappings
315
Altered test logic
318
Testing layer (e.g., "Second Tier")
304
Embedded processing device
308
Altered test logic
318
Embedded Device API
320
Embedded device function library
322
Corresponding Function 1
Corresponding Function 2
Development environment
324

FIG. 3

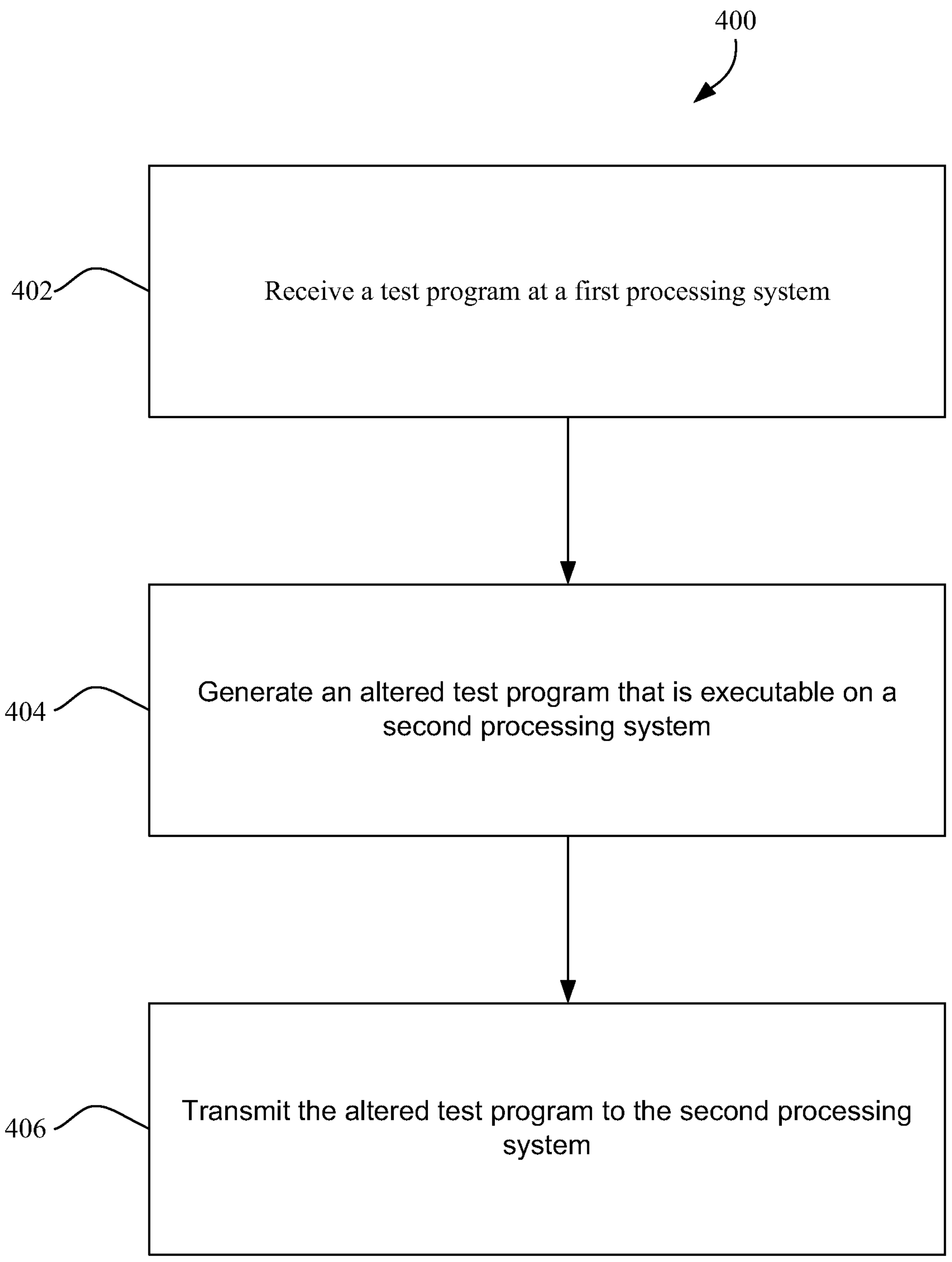
400
402
Receive a test program at a first processing system
404
Generate an altered test program that is executable on a second processing system
406
Transmit the altered test program to the second processing system

FIG. 4

PROGRAMMABLE TEST INSTRUMENT

TECHNICAL FIELD

This disclosure relates generally to a programmable test instrument

BACKGROUND

Automatic test equipment (ATE) plays a role in the manufacture of electronics, such as semiconductor devices and circuit board assemblies. Manufacturers generally use automatic test equipment, or "tester instruments", to verify the operation of devices during the manufacturing process. Such devices are referred to as a "device under test" (DUT) or a "unit under test" (UUT). Early detection of faults eliminates costs that would otherwise be incurred by processing defective devices, and thus reduces the overall costs of manufacturing. Manufacturers also use ATE to grade various specifications. Devices can be tested and binned according to different levels of performance in areas, such as speed. Devices can be labeled and sold according to their actual levels of performance.

SUMMARY

In general, in one aspect, a test instrument includes a first processing system that is programmable to run one or more test programs to test a device interfaced to a test instrument, and that is programmed to control operation of the test instrument, and a second processing system that is dedicated to device testing. The second processing system being programmable to run one or more test programs to test the device, and the first processing system has a first application programming interface (API) and the second processing system has a second API, the first API and the second API being different APIs, the first API and the second API having at least some duplicate functions.

In general, in another aspect, a method performed on a test instrument includes receiving a test program at a first processing system that is programmable to run one or more test programs to test a device interfaced to a test instrument, and that is programmed to control operation of the test instrument, generating, based on the test program, an altered test program that is executable on a second processing system, the second processing system being dedicated to device testing and being programmable to run one or more test programs to test the device, and transmitting the altered test program to the second processing system.

Aspects may include one or more of the following features. Functions of the first API are associated with corresponding functions of the second API. Functions of the first API are configured to call functions of the second API in response to information received by the first processing system. The information includes programming code written using functions of the first API, the functions of the first API calling functions of the second API to configure the programming code to run on the second processing system. Configuring the programming code to run on the second processing system includes generating code using functions associated with the second API in response to use of corresponding functions of the first API. The first processing system is programmable to generate a user interface, the user interface for interacting with the first processing system and for interacting with the second processing system through the first API and the second API.

The first processing system is programmable to generate a user interface, the user interface for interacting with the second processing system through the second API. The first processing system is programmed to run one or more test programs to test the device interfaced to the test instrument, and the second processing system is not programmed to run one or more test programs to test the device. The first processing system is not programmed to run one or more test programs to test the device interfaced to the test instrument, and the second processing system is programmed to run one or more test programs to test the device. The test instrument further includes one or more ports configured to mate to the device, the first processing system is configured to send test data to the device through the one or more ports, or the second processing system is configured to send test data to the device through the one or more ports.

The first processing system has a first application programming interface (API) and the second processing system has a second API, the first API and the second API being different APIs, the first API and the second API having at least some duplicate functions. The test program includes one or more calls to functions of the first API. Generating the altered test program includes identifying one or more corresponding functions of the second API that correspond to the one or more functions of the first API. Generating the altered test program includes replacing the one or more calls to functions of the first API with one or more calls to corresponding functions of the second API. The altered test program is executed on the second processing system to test the device. A development environment associated with the second processing system is provided that allows the altered test program to be modified at the second processing system.

Two or more of the features described in this disclosure, including this summary section, may be combined to form embodiments not specifically described herein.

The systems and techniques described herein, or portions thereof, may be implemented as a computer program product that includes instructions that are stored on one or more non-transitory machine-readable storage media, and that are executable on one or more processing devices. The systems and techniques described herein, or portions thereof, may be implemented as an apparatus, method, or electronic system that may include one or more processing devices and memory to store executable instructions to implement the stated functions.

The details of one or more implementations are set forth in the accompanying drawings and the description below. Other features, objects, and advantages will be apparent from the description and drawings, and from the claims.

DESCRIPTION OF THE DRAWINGS

Fig. 1 is a block diagram of an example test instrument.

Fig. 2 is a block diagram of showing examples of features that may be incorporated into the example test instrument of Fig. 1

Fig. 3 is a block diagram of an example test system.

Fig. 4 is a block diagram of an example tester included in the test system.

DETAILED DESCRIPTION

Described herein is a test instrument that may include multiple processing layers (sometimes referred to as "tiers"). The multiple processing layers may include respective application programming interfaces (APIs) that include function libraries. The function libraries can store various functions that can be called by a program such as a test program or test logic that has been provided to the test instrument at least one level. In some examples, a first layer may programmed to run one or more test programs to test a device interfaced to the test instrument, and may also be programmed to control operation of the test instrument. A second layer of the same test instrument may be dedicated to device testing and may also run one or more test programs to test the device.

The first layer may include a first API and the second layer may include a second API, and the first API and the second API may differ from each other in some way. To allow test programs to be generated at the first layer and executed on the second layer, the first API and the second API may include at least some duplicate functions (e.g., a function of the first API may correspond to a function of the second API). Thus, a test program generated at the first layer (e.g., a test program that includes or calls functions of the first API) can be made compatible with a processing device of the second layer, as at least some of the functions of the first API have counterpart functions associated with the second API.

In some examples, the first layer may store associations or mappings between functions of the first API and corresponding functions of the second API. The stored associations and mappings can be used to alter or translate a test program generated at the first layer into a form that is suitable for execution at the second layer (e.g., by replacing or modifying functions of the first API in the test program to correspond to functions of the second API). The altered test program can then be transmitted to the second layer for execution.

Fig. 1 is a block diagram of an example implementation of the foregoing test instrument 100. In Fig. 1, test instrument 100 includes a three-tiered processing system. However, in other example implementations, there may be more, or fewer, tiers. The different tiers of test instrument 100 reflect the relative relationship of the tiers to the DUT. In this example, first tier 101 includes a computer 102. Computer 102 controls various features of test instrument 100, such as communication with an external network. In addition, computer 102 is programmable to perform various testing operations, as described below. Second tier 104 includes one or more processing devices 106 to 108 that are dedicated to testing. For example, processing devices 106 to 108 typically do not perform non-test functions like test instrument control and network communication. Third tier 110 includes logic 111 to 113 that is programmable both to act as an interface to a DUT 115 and to perform one or more test operations on the DUT.

In this example first tier 101, computer 102 includes one or more processing devices, such as one or more microprocessors or a single multi-core microprocessor (not shown). Computer 102 also includes memory (not shown) that stores executable code to control test instrument communication with the external environment, and to perform various "housekeeping" functions to control operation of test instrument 100. For example, computer 102 may be responsible for exchanging communications between the test instrument and one or more external entities over a network interface, scanning the test instrument for malware, memory management, power control, and other functions that are not specifically related to testing the DUT.

Computer 102 is also programmable to perform test operations on a DUT (e.g., 115) interfaced to test instrument 100. The test operations may include, but are not limited to, testing bus speed, reaction time, or any other appropriate operational aspects of the DUT. In general, the testing that is performed is dependent upon the type of device being tested, and the information sought during testing.

One or more test programs may be loaded into memory on computer 102, and executed by processing device(s) in computer 102 in order to perform the testing. While performing testing, computer 102 may continue to perform other functions, such as those described above, to keep test instrument 100 operational. Consequently, the test latency (e.g., the amount of time between the start of a test and receipt of test results) can be on the order of milliseconds. Furthermore, timing variability can also be affected by the computer 102 performing system functions, (e.g., virus scanning) during one run of a test but not during another. This could result in consecutive steps of a test sequence executing more closely in time when the virus scan is not running relative to when the virus scan is running. This is but an example of the test latency. In different systems, numerous factors may have an effect on test latency, such as the speed of the processing device(s) in computer 102, the amount of memory available in computer 102 to run the test programs, a division of attention among processing devices, .and so forth.

A possible advantage of performing testing via computer 102 relates to development costs of test programs. More specifically, computer 102 may run a windowing, or other relatively user-friendly, operating system. Tools available for development of test programs on such an operating system are typically widely available, and generally familiar to test program developers. As a result, the cost of developing test programs on computer 102, to run on computer 102, can be less than the cost of developing test programs to run on the other tiers of the multi-tiered architecture. This generalization, however, may not apply in all cases.

In this example, second tier 104 includes multiple embedded processing devices 106 to 108. Here, three embedded processing devices are shown; however, test instrument 100 may include any appropriate number of embedded processing devices, e.g., one, two, four, five or more. These processing devices are embedded in the sense that they are incorporated into test instrument 100; and are dedicated to performing test functions (e.g., to testing DUTs interfaced to test instrument 100). Embedded processing devices 106 to 108 typically are not responsible for test instrument operations like the "housekeeping" operations described above that are performed by computer 102. However, in some implementations, embedded processing devices 106 to 108 may be programmed to perform one or more such operations, or other operations not specifically directed to testing DUTs.

Each embedded processing device 106 to 108 may include, e.g., a microcontroller or a microprocessor having a single core or multiple cores. Each microprocessor is programmable, either directly or via computer 102. For example, a user of test instrument 100 may interact with the operating system of computer 102 to program an embedded processing device 106. Alternatively, there may a direct interface, e.g., hardware or software, through which each embedded processing device may be programmed. Programming, in this context, refers to storing one or more test programs onto a respective embedded processing device, which can be executed on that embedded processing device to test a DUT.

As shown in Fig. 1, each embedded processing device is interfaced to computer 102 and to respective programmable logic (in this example, a field programmable gate array (FPGA)). As explained below, each FPGA acts as an interface to a separate DUT (not shown) or to a portion of a single DUT (e.g., a bus

122, 123, 124 on that DUT, as shown) for testing. Accordingly, in this example, each embedded processing device may be programmed with a test program designed specifically for the corresponding DUT, or portion thereof being tested. As noted, an appropriate test program may be loaded directly into the embedded processing device or it may be loaded via computer 102. Each embedded processing device may execute its own test program separately, and concurrently with, other embedded processing devices. In some implementations, there may be coordination among the embedded processing devices as to how their respective test programs are to be executed. Such coordination maybe implemented by the embedded processing device themselves or by computer 102. In some implementations, the coordination may involve devices at different tiers of the architecture. In some implementations, the different embedded processing devices 106 to 108 may implement different portions (e.g., modules) of the same test program, with or without appropriate coordination.

A possible advantage of performing testing via an embedded processing device relates to test latency and timing variability. More specifically, because the embedded processing devices are dedicated to testing, their resources are not typically taxed by other tasks. As a result, testing latency can be less than that achieved by computer 102, and timing variability can be reduced. For example, test latency for an embedded processing device can be on the order of microseconds. This, however, is but an example of embedded processing device test latency. In different systems, numerous factors may have an effect on test latency, such as processing device speed, the amount of memory available to run the test programs, and so forth. Accordingly, the foregoing generalization may not apply in all cases.

Furthermore, tools are available for development of test programs on the embedded processing devices. As a result, the cost of developing test programs targeting an embedded processing device, to run on an embedded processing device, can be less than the cost of developing test programs to run on hardware, such as an FPGA. In some examples, cost issues can relate to the commonness of the skill sets need to program in the various tiers, and to the maturity of the respective development environments. Programming for the embedded tier versus the system tier may require more advanced SW skills. These skills may be less common and more expensive to develop/outsource. System tier development tools are quite refined and powerful relative to the corresponding tools for the embedded tier. This may reduce the cost of development for the system tier (all other things being equal). In addition, the cost of debugging test programs on an embedded processing device can be higher than the cost of debugging those tests on the system processing device for similar reasons. The debugging tools available on the system processing device may be less expensive, more powerful, and more familiar than similar tools available on the embedded processing devices.

Third tier 110 includes programmable logic, e.g., FPGAs 111 to 113, although other types of programmable logic may be used in lieu of FPGAs. Each FPGA is configured by loading a program image into the FPGA. This program image is referred to as an "FPGA load". In this example, each FPGA is configured to act as an interface between a DUT, or portion thereof (e.g., a DUT bus) and test instrument 100. For example, an FPGA may specify a port width, port speed (e.g., 10MHz to 400MHz), the number of input ports, the number of output ports, and so forth.

First tier 101 computing device(s) (e.g., computer 102) and second tier 104 computing device(s) (e.g., embedded processing devices 106 to 108) access DUT 115 through third tier 110. For example, as shown in Fig. 1, each embedded processing device may communicate with DUT 115 through a corresponding FPGA. Computer 102 may communicate with DUT 115 through one or more FPGAs, depending upon which DUT, or portion of a DUT, is being currently tested. In some implementations, each interface implemented by an FPGA is programmable. In other implementations, the interface implemented by each FPGA is static (e.g., not programmable).

Each FPGA may also be configurable to perform one or more tests on a corresponding DUT, or portion thereof to which the FPGA is interfaced. For example, the FPGA load for each FPGA may include one or more test routines that are run by the FPGA to test various aspects of the DUT. As above, the routines that are implemented depend upon the device being tested, and the information sought during testing. Test routines run by each FPGA may be run independently of other test routines run by other FPGAs, or there may be coordination among the various FPGAs. Each FPGA may execute its own test routine separately, and concurrently with, other embedded processing devices. In some implementations, there may be coordination among the FPGAs as to how their respective test programs are to be executed. Such coordination may be implemented by the FPGAs themselves, by their corresponding embedded processing devices, or by computer 102. In some implementations, the coordination may involve devices at different tiers of the architecture. For example, computer 102, in concert with embedded processing devices 106 to 108, may coordinate operation of respective FPGAs 111 to 113. In some implementations, the different FPGAs may implement different portions (e.g., modules) of the same test routine, with or without appropriate coordination.

A possible advantage of performing testing via an FPGA relates to test latency and to reducing timing variability. More specifically, because the FPGAs are hardware devices, they are able to run at higher speeds than the test routines programmed into either the embedded processing devices 106 to 108 or computer 102. As a result, testing latency can be less than that achieved by embedded processing devices 106 to 108 or computer 102. For example, test latency for an embedded processing device can be on the order of nanoseconds. This, however, is but an example of FPGA test latency. In different systems, numerous factors may have an effect on test latency. Accordingly, the foregoing generalization may not apply in all cases.

In some implementations, testing may be performed exclusively by one tier or another of the architecture. For example, computer 102 may be programmed to run one or more test programs to test a DUT, while devices on other tiers of the architecture do not perform DUT tests. Embedded processing devices 106 to 108 may be programmed to run one or more test programs to test a DUT, while devices on other tiers of the architecture do not perform DUT tests. FPGAs 111 to 113 may be configured to run one or more tests on the device, while devices on other tiers of the architecture do not perform DUT tests. Devices that are not performing tests are not necessarily dormant during this time. For example, computer 102 may continue to perform the housekeeping operations described above; the FPGAs may continue to route data to/from the DUT (i.e., to act as interfaces to the DUT); and the embedded processing devices may continue to be active in coordination or other communication (e.g., transmitting test results from the FPGAs to computer 102).

In other implementations, testing may be performed by different tiers of the architecture concurrently or in concert. For example, two or more of computer 102, embedded processing devices 106 to 108, and FPGAs 111 to 113 may act in coordination, at the same time or within the same test sequence, to perform one or more test operations on a single DUT or on multiple DUTs. To effect such coordination, appropriate programming is loaded into computer 102 and/or embedded processing devices 106 to 108, and/or an appropriate image is loaded into the FPGAs. By way of example, a first test may be performed on a DUT by computer 102; a second test may be performed on the DUT by embedded processing device 106; and a third test may be performed on the DUT by FPGA 111. The first, second and third tests may be separate tests, or part of the same test sequence. Data from the first, second and third tests may be combined, e.g., in computer 102, and processed to obtain the appropriate test results. These test results may be sent to an external computer (not shown) for analysis and reporting.

Any tier of the architecture or another (e.g., third party) computer (not shown) may perform the coordination.

In implementations where one or more tiers of the architecture have not been programmed, the unprogrammed tiers may be bypassed (at least as far as their test functionality is concerned). The unprogrammed tiers may be pre-programmed or pre-configured to perform various functions, such as those described above relating to programming and communication among the tiers and with an external network.

Devices at the various tiers may be programmed or configured in real-time. In this context, "real-time" includes programming at test time or shortly before test time. That is, the test instrument need not come pre-programmed with test programs that are to be run on a DUT. Those test programs may be incorporated into the instrument at the appropriate time. Existing test programs on the test instrument may likewise be replaced with new test programs, as appropriate.

Fig. 2 shows a test instrument 200 that includes a multi-tiered architecture. In the example of Fig. 2, the test instrument 200 includes a system layer 202 (e.g., the "first tier") and a testing layer 204 (e.g., the "second tier). However, in other example implementations, there may be more, or fewer, tiers. As discussed above, the different tiers of the system 200 reflect the relative relationship of the system layer 202 and the testing layer 204 to a DUT (not shown). In this example, the system layer 202 can provide similar functionality to the first tier 101 shown in Fig. 1, and may include a system processing device 206. For example, the system processing device 206 may control various features of the test instrument 200, such as communication with an external network or other general computing tasks (e.g., controlling programs unrelated to testing DUTs, such as email programs and word processing programs). In addition, the system processing device 206 can be programmable to perform various testing operations, as described below.

The testing layer 204 includes an embedded processing device 208 that is dedicated to one or more specialized tasks, such as device testing. While the testing layer 204 may include more than one embedded processing device (e.g., as shown in the second tier 104 of the test instrument 100 (Fig. 1)), only one embedded processing device is shown for the sake of simplicity. In some examples, the embedded processing device 208 does not perform non-test functions like test instrument control and network communication. The test instrument may also include one or more additional layers, such as the third tier 110 shown in Fig. 1, which may in turn communicate with one or more DUTs.

The system layer 202 also includes a system application programming interface (API) 210. In general, an API is a set of data structures, functions, protocols, routines and tools for accessing and interacting with an application (e.g., software running on one or more of the system processing device 206 and the embedded processing device 208). An application programming interface may be language dependent or language independent. A language dependent API is available only in a particular programming language, while a language independent API is not bound to a particular language, system or process. In some examples, language independent APIs can be accessed from (e.g., have their functions called using) more than one programming language. The system API 210 includes a system device function library 212 that stores a number of functions associated with the system API 210 (e.g., Function 1 and Function 2). A function (e.g., Function 1) may, when called, cause the system processing device 206 to perform one or more operations. For example, a function (e.g., Function 1) can cause a processing device to access a register associated with an FPGA (e.g., an FPGA associated with the third tier 110). Other functions may access a power supply or temperature sensor (e.g., inside the instrument) to monitor the status of the instrument.

In some examples, a user 201 may wish to develop test logic that can be used by the test instrument 200 to test a DUT. The system layer 202 may provide a development environment that includes, or has access to, the system device function library 212. While some system layers are not typically able to provide a development environment that provides the ability for the user 201 to create test logic (e.g., because test logic is executed on the embedded processing device 208 in the testing layer 204, and functions syntax and availability may vary between layers of the test instrument 200), the techniques described herein allow the user 201 to create test logic at the system layer 202 that can be executed at the testing layer 204 to test DUTs connected to the test instrument 200 (e.g., DUTs connected to a third layer, such as the third tier 110).

In some examples, to allow for test logic creation at the system layer 202, the system layer 202 and the testing layer 204 can be provided with respective APIs that include functions that correspond to one another. For example, the embedded device API 214 can include an embedded device function library 216 that stores functions (e.g., Corresponding Function 1 and Corresponding Function 2) that correspond to functions of the system device function library (e.g., Function 1 and Function 2, respectively). Such an arrangement allows, for example, the user 201 to develop test logic at the system layer 202 that can be executed by the embedded processing device 208 associated with the testing layer 204. For example, because at least some of the functions of the embedded device function library 216 have respective counterpart functions in the system device function library, test logic developed at the system layer 202 can include functions calls that properly call the desired functions of the embedded device function library 216. As a result, test logic can be created that can be executed at least in part on the system processing device 206 and that properly calls functions of the testing layer 204 (or other layers) when necessary. For example, some functions of system device function library 212 can be wrappers to their embedded device function library 216 counterparts.

Providing similar functions at both the system layer 202 and the testing layer 204 allows developers who are unfamiliar with, for example, developing test logic at the test level 204, to develop usable test logic at the system layer 202 (e.g., in a development environment that the developer may be more familiar with). However, in some examples, some test logic or portions thereof must be developed at the testing layer 204 due to performance demands of certain test requirements, such as latency or timing variability constraints.

Fig. 3 shows a test instrument 300 that includes a system layer 302 and a testing layer 304. Like the test instrument 200 discussed above with regard to Fig. 2, the test instrument 300 includes features that allow a user 301 to generate test logic (e.g., test logic 316) that can be transferred or ported to the testing layer 304 for execution. For example, the user 301 may develop test logic 316 within a development environment 313 associated with the system layer 302. In some examples, the development environment 313 can run on a layer or machine different than the system layer 302. While the test logic may include functions (e.g., Function 1 and Function 2) provided by a system device function library 312 associated with a system API 310, the user 301 may desire to have some or all of the test logic 316 be executed on an embedded processing device 308 associated with the testing layer 304.

To facilitate the execution of test logic 316 that uses functions from the system function library 312 (e.g., and not the embedded device function library 322), the system layer 302 may use a test logic translation engine 314 to alter, translate, or otherwise modify the test logic 316 into a format that can be executed by the embedded processing device 308. In some examples, the test logic translation engine 314 includes one or more function mappings 315 that specify associations between a function of the system function library 312 and a corresponding function of the embedded device function library 322. Using the mappings 315, the test logic translation engine can modify the test logic 316 to provide altered test logic 318 that is suitable for execution on the embedded processing device 308. In some examples, the test logic translation engine 314 can use the mappings 315 to replace functions (e.g., Function 1) associated with the system function library 312 with its counterpart function in the embedded device function library 322 (e.g., Corresponding Function 1).

After the test logic translation engine 314 has generated the altered test logic 318, the system processing device 306 of the system layer 302 can provide the altered test logic 318 to the embedded processing device 308 for execution. Before the embedded processing device 308 executes the altered test logic 318, the testing layer 304 may perform one or more tasks, such as error checking, compilation, or other pre-processing operations (e.g., testing layer 304 (and specifically the embedded device function library 322) can error check the altered test logic 318). In some examples, some of these checks are of the behavior of this test logic as it runs, and some of these checks are of the altered test logic 318 binary itself. These checks can be performed, for example, when the altered test logic 318 is loaded. Furthermore, in some examples, it may be desirable to modify the altered test logic 318 prior to execution on the embedded processing device 308 (e.g., test performance requirements, such as latency or response time specifications, may require further modifications to be made to the altered test logic 318 at the testing layer 304). To facilitate further modification of the altered test logic 318 at the testing layer 304, the testing layer 304 may also include its own development environment 324 through which the user 301 may access and alter the altered test logic 318 that has been ported over from the system layer 302. In some examples, the first tier may have a larger latency and timing variability associated with it than the second tier. Furthermore, in some cases, action by the test logic can trigger a UUT response that completes faster than the first tier's timing is latent or variable. In these cases, some SW logic optimizations might not produce a measurable performance improvement in the first tier yet might produce a dramatic improvement in the second tier. In some examples, the system layer 202 (or 302) may also provide a user interface 203 (Fig. 2) that can be used to interact with one or both of the system API 210/310 and the embedded device API 214/314. For example, the user interface can act as a common interface to both development environments 313 and 324, which may allow the user 301 to modify test logic at either the system layer 302 or the testing layer 304. Thus, the techniques described above can allow the user 301 to generate early test logic at the system layer 302 (e.g., in a development environment that is familiar to the user 301), and the test logic can then be translated into altered test logic that is ported to the testing layer 304 for further modification and/or execution. These techniques can also allow user 301 to debug the early test logic at the system layer 302 using more powerful and familiar debugging tools. These techniques can be used instead of, or in combination with, the techniques discussed above with regard to Fig. 2. For example, the existence of corresponding functions in function libraries of the system and testing layers allows the test logic translation engine 314 (and its associated mappings 315) to identify appropriate functions at the testing layer 304 that should be substituted for system layer functions in the test logic 316.

Fig. 4 illustrates a process 400 that allows a test program to be created at a first processing system and to be modified for eventual execution at a second processing system. The process 400 can be executed, for example, on a test instrument. A test program is received at a first processing system (402). For example, a user can generate a test program using a development environment associated with the system layer 302. The user may also upload an existing test program 302 to the first processing system via a network connection or a suitable storage device (e.g., a portable storage device, such as a universal serial bus (USB) thumb drive). In some examples, the first processing system is programmable to run one or more test programs to test a device interfaced to a test instrument, and is programmed to control operation of the test instrument.

An altered test program is generated that is executable on a second processing system (404). In some examples, the second processing system is dedicated to device testing and is programmable to run one or more test programs to test the device. The first processing system may have a first application programming interface (API) and the second processing system may have a second API, where the first API and the second API are different APIs, and the first API and the second API have at least some duplicate functions. While the test program may include one or more functions of the first API, generating the altered test program may include identifying one or more functions of the second API that correspond to the one or more functions of the first API. Generating the altered test program may also include replacing the one or more functions of the first API with the one or more corresponding functions of the second API.

The altered test program is transmitted to the second processing system. For example, after the altered test program has been generated in a form that is suitable for execution on the second processing system, the altered test program can be provided to the second processing system. The second processing system may also provide a development environment that allows the altered test program to be modified at the second processing system.

While some of the foregoing examples describe providing altered test logic to one embedded processing device of the testing layer, the system layer can provide altered test logic to multiple embedded processing devices of the testing layer. In some examples, more than one embedded processing device can be associated with one or more shared APIs. Similarly, one or more system processing devices can be associated with one or more shared APIs.

The foregoing describes performing testing using a system processing device, embedded processing devices, or programmable logic. However, testing, as described herein, may be performed using a combination of system processing device, embedded processing devices, or programmable logic. For example, each of these different elements may run on or more test programs simultaneously to test the same device or portion thereof. Likewise, these different elements may coordinate testing so that, e.g., a system processing device (e.g., 102 of Fig. 1) performs a first part of a test sequence, an embedded processing device (e.g., 106 of Fig. 1) performs a second part of the same testing sequence, and programmable logic (e.g., FPGA 111 of Fig. 1) performs a third part of the same testing sequence. Any appropriate coordination may take place between the different programmable elements of the test instrument described herein.

Furthermore, in some implementations, a tier of processing may be circumvented. For example, testing may occur using a system processing device (e.g., 102) and programmable logic (e.g., FPGA 111), but not an embedded processing device. In such implementations, communications between the system processing device and the programmable logic may pass through an embedded processing device or bypass the embedded processing device tier altogether.

In some implementations, there may be more than three tiers of processing devices. For example, there may be two tiers of embedded processing devices (resulting, e.g., in four tiers total). For example, a single embedded processing device may be used to coordinate testing of a single device, and different embedded processing devices (under the direction of that single embedded processing device) may be used to test different aspects or features of that single device.

In some implementations, one or more tiers of processing devices may be eliminated from the system of Fig. 1. For example, some implementations may not include a tier of embedded processing devices. In such example systems, there may be only a system processing device (e.g., 102 of Fig. 1) and programmable logic (e.g., FPGAs 111 to 113. In this regard, any appropriate combination of tiers may be employed in the test instrument described herein.

In some implementations, the system processing device (e.g., 102 of Fig. 1) may be external to the test instrument. For example, an external computer may be employed to control operations of the test instrument, and may interact with embedded processing device(s) and programmable logic on the test instrument in the manner described herein. In other implementations, the system processing device may be part of the test instrument or remote from the test instrument (e.g., connected to the test instrument over a network).

In some implementations, the programmable logic may be replaced with non-programmable logic. For example, rather than using an FPGA, one or more application-specific integrated circuits (ASICs) may be incorporated into the test instrument in place of, or in addition to, the programmable logic described herein.

The functionality described herein, or portions thereof, and its various modifications (hereinafter "the functions"), are not limited to the hardware described herein. All or part of the functions can be implemented, at least in part, via a computer program product, e.g., a computer program tangibly embodied in an information carrier, such as one or more non-transitory machine-readable media, for execution by, or to control the operation of, one or more data processing apparatus, e.g., a programmable processor, a computer, multiple computers, and/or programmable logic components.

A computer program can be written in any form of programming language, including compiled or interpreted languages, and it can be deployed in any form, including as a stand-alone program or as a module, component, subroutine, or other unit suitable for use in a computing environment. A computer program can be deployed to be executed on one computer or on multiple computers at one site or distributed across multiple sites and interconnected by a network.

Actions associated with implementing all or part of the functions can be performed by one or more programmable processors executing one or more computer programs to perform the functions All or part of the functions can be implemented as special purpose logic circuitry, e.g., an FPGA and/or an ASIC (application-specific integrated circuit).

Processors suitable for the execution of a computer program include, by way of example, both general and special purpose microprocessors, and any one or more processors of any kind of digital computer. Generally, a processor will receive instructions and data from a read-only memory or a random access memory or both. Components of a computer include a processor for executing instructions and one or more memory devices for storing instructions and data.

Components of different embodiments described herein may be combined to form other embodiments not specifically set forth above. Components may be left out of the circuitry shown in Figs. 1 to 4 without adversely affecting its operation. Furthermore, various separate components may be combined into one or more individual components to perform the functions described herein.

Other embodiments not specifically described herein are also within the scope of the following claims.

What is claimed is: