Infrastructure IP for Programming and Test of in-system Memory Devices

CJ Clark and Mike Ricchetti
Intellitech Corporation, 70 Main Street, Durham, NH 03824
cjclark@intellitech.com and miker@intellitech.com

Abstract

Today's efforts to reduce both system manufacturing costs and in-the-field in-system upgrades requires new flexible and cost effective methods of testing complex memories and programming non-volatile memories. FLASH memory, Serial EEPROM, SDRAM, NAND memory, FCRAM and DDRRAM, all present problems for efficient in-system programming and test in systems with limited physical access. Reduced contact methods based on IEEE 1149.1 for testing PCBs and Systems rely more and more on in-chip Design for Test (DFT).

The IC and SoC designer can add differentiation and value not just by incorporating design features and Design-for-Test for the IC, but by considering the target system that the IC/SoC may be used in. This practice is, of course, common in vertically integrated companies, but not standard practice by independent silicon vendors. The silicon vendor that can lower not only his customer's cost of the part, but also the entire cost of using the silicon can gain a competitive advantage.

This paper describes a patented architecture that is applicable to any silicon that is coupled to any of the aforementioned memory types. The example used in the paper, however, is for an external FLASH memory. Traditional methods of programming the FLASH and their drawbacks are described here and are compared to the Fast Access Controller (FAC) architecture.

Introduction

FLASH memory is commonly used in processor and microcontroller based designs to store the system's firmware. Modern day PCBs with mezzanine cards, and complex multi-board systems, may have multiple processors each with their own local FLASH memory for storing program code. FPGA logic may also be loaded from a dedicated FLASH device, so a complex multi-board system may typically have 3-4 different FLASH devices, with more for larger systems. The re-programmability of FLASH offers the flexibility for upgrades and bug fixes when product designers adopt a method for in-system non-volatile memory programming. In-system programming also enables the FLASH devices to be programmed with the latest firmware release during the manufacturing process. This avoids keeping an inventory of pre-programmed non-volatile memory parts.

There are many ways to program FLASH memory (stand-alone programmers, in-line programmers, in-circuit emulation equipment, in-circuit "pogo-pin" testers, and boundary-scan testers), each method has certain drawbacks such as cost and slow FLASH programming times. Each method is also just a point solution targeted for only one phase of the product's life, for instance in-line programmers cannot easily be used during engineering prototyping, emulation equipment can't easily be used in production test. If a system is being designed to enable remote software and FPGA logic updates, yet, another solution must be devised to allow each individual FLASH to be updated when the product is in the field. In some cases, 3 or 4 of these programming methods are used, each one costing more in equipment and engineering resources to support over each phase of the product's life.

Consequently, a structured access method for all system non-volatile devices is needed in order to reduce costs and provide fast, flexible, upgrades and re-configuration. The patented FAC architecture [2], [3] enables in-system programming of volatile and non-volatile memory devices as fast as off-board, or direct access, programming techniques. More importantly, it provides a unified method in all environments and throughout the product's life cycle.

Current FLASH Programming Methods

The following subsections briefly describe traditional method used to program FLASH memory devices.

Gang and In-Line FLASH Programmers

The major problem with using gang FLASH-programming stations is that this method increases inventory, adds to manufacturing complexity and cost, and requires early software code freezes. Prior to production, software engineering must finalize code so FLASH parts can be programmed and inventoried. The FLASH parts have to be manually loaded on the programming station, the FLASH is then programmed, then the newly programmed parts have to be inventoried and then eventually inserted onto a board. If a last minute software change is needed, this process needs to be repeated. This has forced some OEMs and Contract Manufacturers (CMs) to program FLASH devices using 'inline programmers' during manufacturing to achieve automatic device handling and the elimination of inventories of programmed parts. However, only very high volume products can justify the cost of an inline programmer. The higher the volume, and the larger the FLASH device (requiring more FLASH to be programmed in parallel), the more costly the inline programmer solution is. Multiple inline programmers may even be needed to allow for certain types of line balancing typically done with multiple assembly and test lines. Other disadvantages include; new techniques must be developed to re-program the FLASH for updates after the PCB leaves the manufacturing floor; opens testing to the pre-programmed FLASH device requires additional test engineering time, and finally pre-programmed FLASH will inevitably be placed on PCBs with manufacturing defects, which increases overall costs.

Using ICT for FLASH Programming

OEMs have also leveraged In-Circuit Testers (ICT) to program FLASH parts on-board. This method allows testing the PCB interconnects first, i.e. prior to FLASH programming, and provides reasonable programming performance. However, this can greatly impact the design as it requires the designer to add test points for direct access to all FLASH device pins. With the increased use of BGA technology, adding test points or enlarging VIAs may not be possible without delaying design schedules, increasing layout area or impeding critical timing paths. Like, inline FLASH programming, ICT is not a highly flexible FLASH programming platform because programming FLASH using ICT cannot be done outside of the manufacturing facility, for example in the field. Furthermore, it requires that a new test program be developed and re-compiled each time there is a change in the FLASH memory contents. Some ICT can program FLASH memory with boundary-scan based software, (see boundary-scan FLASH programming below) while this removes the need for the test points, this approach is the most costly of any of the methods described. Since programming large FLASH through boundary-scan can take minutes, the ICT cannot be testing any other PCBs during that time, the throughput and tester utilization suffers. Allocating the long programming times onto high dollar per hour test platforms such as ICT increases costs and affects throughput, since ICT is a "one-at-a-time" approach.

FLASH Programming Using Emulation Tools

Some FLASH devices can be programmed in-system through an adjacent system processor (CPU, Micro Controller or DSP). This method requires the use of an in-circuit emulation (ICE), or background debug mode (BDM) tool and hardware pod. The ICE/BDM downloads a small FLASH loader program via the processor debug port to the processor's RAM, and then the processor delivers the program data to the FLASH. In the lab, this method provides good programming performance and is relatively easy to use for embedded software developers. However, new processors, even those in the same family require new loaders, new FLASH devices require new algorithm files to program the FLASH and in some cases the FLASH programming loader is design dependent requiring changes to support other designs that might address the FLASH differently. Finally, setup and verification must be done by engineers familiar with both the software and the design, such as firmware engineers, and typically is not practical for production personnel to support when something goes wrong.

In a manufacturing environment using an ICE/BDM pod to program FLASH has several disadvantages. First, the ICE/BDM hardware and software must be integrated with manufacturing test equipment and manufacturing failure tracking database and software. Today's CM's and EMS' companies need 'standardized' production lines that can quickly handle switching to different customers. Integrating a ICE/BDM for a particular product can add to this the complexity. Since the emulation pod is only used for this function it increases test equipment costs and requires integration of diagnostic information with other test equipment, or adds another step in the process since the ICE based solution can not perform other manufacturing tests (such as IC to IC interconnect tests, that would require the addition of boundary-scan tools). A further disadvantage for manufacturing with this method is that it requires the board to have a working processor and some defect-free scratch memory, so that a processor-specific FLASH loader program can run. And with no software loaded in the FLASH, testing the processor connections to other devices and the FLASH would first need to be done through another means, such as boundary-scan or ICT. Otherwise, when the ICE/BDM pod is used, the small FLASH loader may fail to load and without diagnostics to help it would be difficult to pinpoint the failure (which can be one of dozens of connections the CPU needs in order to execute the loader program).

Further problems exist in that many ICE/BDM based emulation pods require that the CPU be on a separate scan-chain from other 1149.1 devices. Multiple, separate, scan-chains on a PCB can add to the complexity of the PCB scan-chain design itself, manufacturing interfaces and access during field updates. Deploying ICE/BDM equipment to a world-wide distribution of field service personnel for the purpose of updating just the FLASH connected to the microprocessor (and using a separate tool for serial EEPROM for instance) in the field is not cost effective.

FLASH Programming with Boundary-Scan

Weaknesses with the above FLASH-programming techniques have led to increased use of the standard IEEE 1149.1 [1] test infrastructure, using the EXTEST boundary scan instruction to access the FLASH devices for programming. This technique has limited impact on the design and only requires that the FLASH memory's address, data and control signals be directly connected to an IEEE 1149.1 compliant device, so that the device's boundary register can be scanned to perform FLASH write sequences in EXTEST. Although there is little design impact, in-system programming of FLASH devices using this method has serious performance issues. Given the large pin count of today's boundary scan devices, and the number of boundary-scan devices used on a board, the length of the scan path becomes a major limiting factor in FLASH programming times when using this method. Other factors such as the maximum TCK frequency achievable on the PCB, the amount of program data to be written, the number of scans the FLASH device requires for each write cycle, and the "burn time" requirements of the FLASH device, also impact the total programming time. While it has been suggested by some to add direct physical access to the FLASH device's Ready/Busy pins, if the time to scan a long boundary-scan chain is longer than the minimum FLASH "ready" time, no benefit is gained. The majority of the time in programming the FLASH is shifting the data through the long boundary-registers. Incorporating physical access to the FLASH write enable (WE) input *can* cut the programming time almost in half, however in practice reducing the programming time by half will not result in an acceptable improvement. Programming times in the tens of minutes are still not practical in a manufacturing environment. In the field, especially in remote areas of the world, it may not be economical to supply all field personnel with high-speed boundary-scan controllers capable of delivering data at 10-20Mhz.

Furthermore, the added cost of carrying the direct access through pins and routes in a passive backplane or multi-PCB design cannot be overlooked. In complex multi-PCB systems, adding physical access to these pins requires additional design time and adds components and design complexity. "Buffered Factory programming" techniques implemented in the FLASH itself offer no speed up in FLASH programming when the EXTEST technique is used since each word or byte must still be scanned in through the boundary-scan chain.

In-system Programming using the FAC

In order to overcome the obstacles presented above, the internal development for a Fast Access Controller began in 1997, followed by formal patent write-up at the end of 1999 and provisional patent filing with the USPTO in March of 2000. The FAC is an infrastructure IP block [5] designed to provide for high-speed, high-throughput, in-system configuration and test of memories. It leverages the test infrastructure of IEEE Std. 1149.1 to enable in-system programming of FLASH and other non-volatile memory devices, as fast as off-board or direct access programming techniques. The FAC can achieve optimal programming throughput of FLASH devices, even with lower test clock rates (<3Mhz). The scan length or number of 1149.1 devices in the boundary-scan chain of the PCB does not affect the FAC architecture. As a result, the performance issues associated with 1149.1-based in-system FLASH programming, using the EXTEST approach, are eliminated.

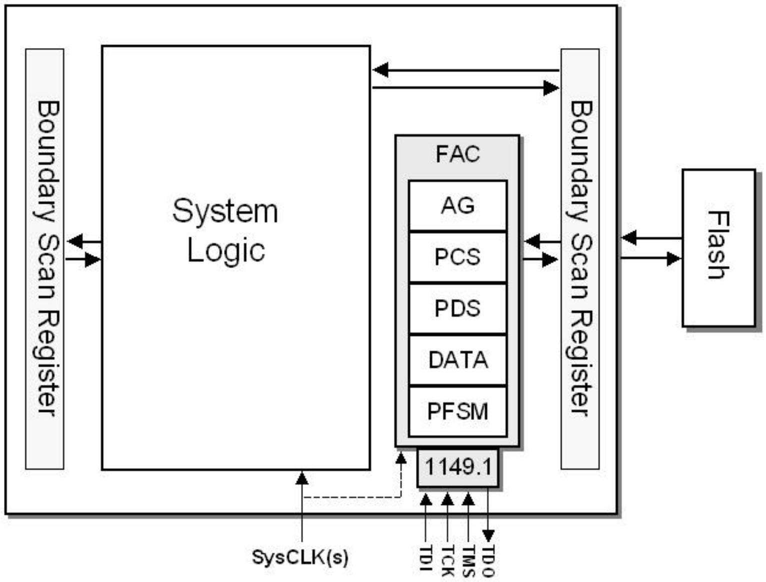

Figure 1. IC with Boundary Scan and Fast Access Controller (FAC)

The performance achieved by the FAC is made possible by the novel features of the architecture, which are described below. These features combine to minimize the number of scan operations and serial scan data required during FLASH programming or memory testing. The memory protocols of the FAC are programmable in-system, through PC based boundary-scan tools. The FAC is also customizable in-system, which allows it to support access to a wide variety of memory devices such as NOR FLASH, NAND FLASH, EEPROM, Serial EEPROM, SRAM, SDRAM and DDRAM and other memory devices. Flexibility is key to the design as when the FAC is implemented in a CPU or ASIC, typically the target memory type or memory manufacturer is not known at design time. The FAC enables an IEEE Std. 1149.1 bus to be used as a central bus for in-system configuration of all on-board non-volatile memory devices. In addition, it provides a single non-volatile programming method that can be used in the field, during prototype bring-up, and for manufacturing test and configuration.

The FAC Architecture

A block diagram of the FAC is shown in Figure 1.
The FAC is comprised of several functional blocks: the Address Generator (AG), the Programmable Control Sequencer (PCS), the Programmable Data Sequencer (PDS), the Data Register (DATA) and the programmable Finite State Machine (PFSM.

In this example implementation of the FAC architecture, the FAC is embedded in an IC, such as an ASIC or FPGA, and interfaces to a FLASH memory device. The interface to the FLASH is through multiplexers that select between the system logic and the FAC. This multiplexing can be integrated into the Boundary Scan Register (BSR), as shown in Figure 1. The FAC also interfaces to other 1149.1 logic, such as the Instruction Register (IR). It may optionally take system clocks as inputs. These system clocks enable the FAC to access a memory device for at-speed test to memories especially SDRAM and DDRRAM. When full speed access is not required, the FAC may operate from only the TCK clock of the TAP.

The FAC operates through specific FAC-based 1149.1 instructions, together with the sequencing of the PFSM. Figure 2 shows the state diagram of the FAC's PFSM. As can be seen, the PFSM is an extended version of the standard IEEE 1149.1 TAP Controller FSM. The PFSM includes a modified DR branch in the state machine, which provides for programmable states. These programmable states are decoded and used to control the operation of the FAC, the FAC registers and the protocols it generates. The FAC branch is taken from the *Select-DR-Scan* state when there is a FAC specific instruction loaded into the IR, i.e. FAC_Op = 1 and TMS = 0. This FAC_Op branch provides the ability to perform protocol sequences that are not possible to perform in the DR branch of the standard 1149.1 TAP controller FSM. Because the PFSM allows programmable states, the placement of the *Capture-DR*, *Update-DR* and *Pause-DR* states in the FAC_Op branch can be determined based on the FAC instruction and its operation. This removes the restrictions of the fixed protocols in the standard TAP controller FSM, allowing for an extended test capability. Nevertheless, the PFSM remains compatible with the standard IEEE 1149.1 architecture. An example of FAC operation for a FLASH_PROGRAM instruction is provided in the next section.

The PFSM states are defined as follows:

- *Enter-FAC.* Enables the FAC.
- *APG1-DR.* Enables FAC operations concurrent with DR shift. May also be used for *Pause-DR*.
- *Update1/Capture1-DR.* Programmable *Update-DR* and/or *Capture-DR* and FAC operation.
- *APG2-DR. E*nables FAC operations concurrent with DR shift. May also be used for *Pause-DR*.
- *Update2Capture2-DR.* Programmable *Update-DR* and/or *Capture-DR* and FAC operation.
- *Exit-FAC.* Disables the FAC.

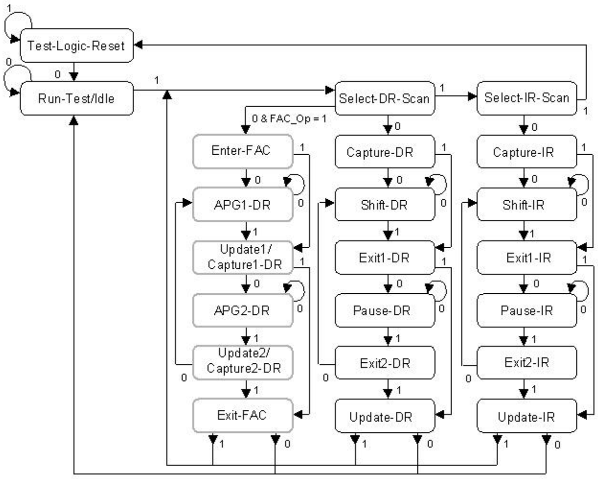

Figure 2. The Programmable FSM of the FAC's TAP Controller

Figure 3 shows a more detailed diagram of the functional blocks in the FAC, and how they interface to each other. The PFSM connects to an IR, to the other FAC blocks, and to the standard 1149.1 DRs of the IC (i.e., the Bypass Register and a BSR, which are not shown in figure 3). The FAC also has a number of DRs, which can be accessed individually or in some implementations all of the registers are linked together except the DATA register which is accessed separately during the reading or writing to the memory. Each of the PCS, AG, DATA and PDS blocks shown in figure 3 has one or more DRs. These are used to load data, address and control protocol into the FAC, either initially or while it is in operation. The DRs may be dedicated registers for use only during FAC operation, or may be shared with the BSR or functional registers (i.e., other internal scan registers) of the IC.

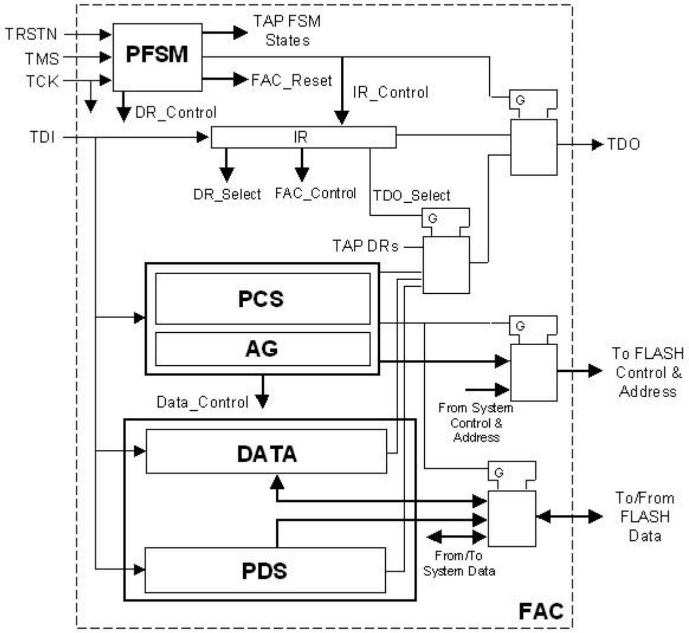

Figure 3. FAC Block Diagram

The FAC blocks receive control inputs from the PFSM and the IR logic. The DR_Select control signals from the IR are used to select the FAC DR to be shifted, while the DR_Control signals from the PFSM are used to control the scan shift operations of the DRs. The FAC_Cntrl signals are also received from the IR logic, which controls and enables FAC operation together with the PFSM states.

During operation of the FAC the PCS provides the FLASH's Control signals via the test multiplexers, shown in figure 3, in a pre-programmed sequence. This control sequence, or protocol, is directly programmable by means of the PCS DR(s). The PCS is enabled to operate during the FAC_Op states of the PFSM. Depending on the FAC instruction loaded into the IR, and the PFSM state, the PCS will be started or stopped, or simply continue to execute its protocol sequence to the circuit connected to the FAC. The PCS and the other functional blocks of the FAC are permitted to execute concurrently with the operations in the FAC_Op branch. For example, the PCS and AG can operate during the *APG1-DR* (*Shift-DR*) state, so that the data DR in the DATA logic can be scanned concurrently while control and address signals are being applied to the FLASH automatically by the PCS and AG. This feature of the FAC architecture provides for optimal scan-based access to the FLASH device.

In addition to sequencing the controls to the FLASH, the PCS also interfaces to the other functional blocks of the FAC, such as the AG, PDS or DATA, in order to sequence them along with the protocol for accessing the FLASH. For example, the PCS may select or control a particular address sequence to be generated by the AG, or a data sequence to be generated by the PDS. During its operation, the FAC logic can be synchronized with either the TCK clock or system clocks, so that it can operate at the full system speed of the memory device being accessed.

The AG logic outputs address sequences through the test multiplexers onto the FLASH's Address bus, as shown in figure 3. By automatically providing address sequences from the AG to the memory device connected to the FAC, as opposed to providing the addresses via the BSR, scan operations do not have to be performed each time a new memory location is accessed. The AG may contain one or more address generation circuits. For example, it may include both an address counter, to sequentially address the FLASH during read/program operations, and an address sequence generator, which is used to generate fixed sequences of addresses for the FLASH's program/erase commands. Each of these generators in the AR logic may have a DR and instructions to scan the DRs allow the AR address registers to be initialized to a starting address if desired. The address generation sequences are selected and controlled by specific FAC instructions, the PFSM and the PCS. For example, in the FLASH implementation of figures 1 and 3, during a read operation controls from the PCS signal the AR to advance to the next memory address to be accessed by the FAC as the read protocol is being sent by the PCS.

The DATA and PDS logic shown in figure 3 provide for read/write operations, and special data sequences, to/from the FLASH device. The DATA register is a DR that can be scanned during FAC operation to scan-in write data, to be written to the memory, or scan-out read data, that has been read from the memory. In the FLASH implementation of figures 1 and 3, the PDS is programmed with data sequences required for the FLASH's program/erase commands. The DATA and PDS logic are controlled based on the instruction loaded in the IR, the PFSM states, and the PCS protocol. For example, during a FLASH_PROGRAM instruction, the DR of the DATA logic is scanned during the *APG1-DR* state, while the PDS outputs data values for a program command sequence to the FLASH data bus. The DATA DR being separate from the PDS DR allows it to be scanned concurrently, while the PDS is outputting data and the PCS is outputting control signals. When read or write data from the FLASH is transferred from/to the DATA DR, it receives the proper control input for Capture and Update operations from the FAC TAP DR_Control signals.

Table 1 illustrates this concurrent operation of the functional blocks in the FAC. The table shows a short sequence of PCS controls and the data and addresses output by the PDS/DATA and AR respectively. As the PCS sequences through the FLASH's CEN, OEN and WEN controls, the AG and PDS output fixed address and data sequences corresponding to a FLASH program command. During this sequence, the DR in the DATA logic can be scanned with write data. When the PCS's CTA and CTD control is 1, it selects the address and write data for the program command to be sourced from the AG's address counter and the DATA's DR, respectively. CTA and CTD are separate for other types of memories where the Address is not delivered in the same sequence as the data as it is with the example FLASH. The signals in the shaded area of Table 1 are for implementing the sequences for use with NAND FLASH devices and DRAM devices. In the FAC implementation Table 1, essentially has equivalent sized scan registers that are filled during FAC initialization with the values to be used for a particular memory. The CMP signal can be used to force a comparison between the data in the data register (the data written) and a subsequent read back of the data from the memory. This should be done with caution as it does not allow 'data uniqueness'. Consider if the device is a EEPROM or SRAM, if data is read back directly after writing then all address lines could be stuck, yet verification would pass as each unique data value would be written over and over to the same address.

FAC Operation

To further illustrate the operation of the FAC, the following steps provide an example of how a FLASH_READ instruction works.

	AG	0x5555	0x5555	0x2AAA	0x2AAA	0x5555	0x5555	AddrCnt	AddrCnt	AddrCnt	AddrCnt
	PDS/ DATA	0x00AA	0x00AA	0x0055	0x0055	0x00AA	0x00AA	DATA	DATA	DATA	DATA
	CMP	0	0	0	0	0	0	0	0	0	1
PCS	CTA	0	0	0	0	0	0	1	1	1	1
	CTD	0	0	0	0	0	0	1	1	0	0
	DDIR	1	1	1	1	1	1	1	1	0	0
	CEN[0]	0	0	0	0	0	0	0	0	0	0
	OEN	1	1	1	1	1	1	1	1	0	0
	WEN	0	1	0	1	0	1	0	1	1	1
	A/D	0	0	0	0	0	0	0	0	0	0
	RAS	0	0	0	0	0	0	0	0	0	0
	CAS	0	0	0	0	0	0	0	0	0	0
	BA[0..2]	0	0	0	0	0	0	0	0	0	0
	CKE	0	0	0	0	0	0	0	0	0	0
	CK	0	0	0	0	0	0	0	0	0	0

Table 1. **Example FAC PCS, AG and PDS/DATA Sequencing**

The read starts with the following FAC initialization steps:

1. Reset the TAP controller. This will also reset the FAC.
2. The PRELOAD instruction and the boundary-register is loaded into the IC with the FAC and other ICs that are present on the PCB as needed.
3. Load the SHIFT_PCS_AR instruction into the IR. Load the PCS’s DR with the read protocol and initialize the AR with a starting address.
3. Load the FAC_READ instruction into the IR.

4. The FAC_READ instruction sets FAC_Op = 1, enabling FAC operation. Subsequent TAP controller protocol is decoded on the FAC_Op branch of the PFSM, it also selects the DR in the DATA block.

After the read instruction is updated, the following steps are taken:

5. Transition the PFSM to *Enter-FAC* to start the FAC operation.
6. The PFSM moves to *APG1-DR* and the DR in the DATA logic is shifted. Concurrently, the PCS and AR apply a read sequence to the FLASH with the first address in the AR address counter DR.

Note that the initial scan-out data that is shifted out of the DR is “don’t care” data, since the first read has not been completed. Scan-in data is not used during reads, so it can be set to all ones or all zeroes data.

7. When the PFSM enters *Update1/Capture1-DR* the last bit of the DATA DR is shifted. During this DR shift, the read cycle to the memory completes.
8. The PFSM transition through Update1/Capture1-DR and entering *APG2-DR* the data read from the FLASH is captured into the DATA DR.
9. The PFSM is moved back to *APG1-DR* and step 1 is repeated, with the data from the first read being scanned out while the next address is read.
10. After the final read address has been shifted out of the DATA DR, the PFSM can move directly from *Update1/Capture1-DR* to *Exit-FAC*, and then back to *Run-Test/Idle*.

When the PFSM enters *Exit-FAC*, it sets FAC_Op = 0 and disables the FAC operation. It should be noted that in step 7, the FAC could wait in *APG2-DR* for additional TCK cycles if “*Pause-DR*” time is required for a particular FAC operation. For example, to account for the burn time when programming the FLASH.

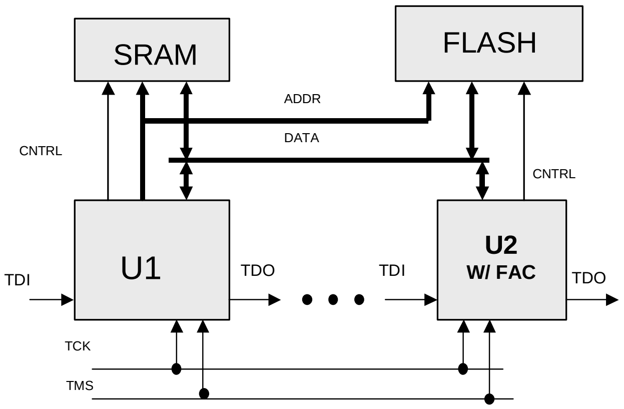

Figure 4.

In order to disable the data bus on the PCB so FLASH programming can begin, certain PCBs will require components on the data bus to be put in EXTEST or CLAMP. If U1 of Figure 4 did not support CLAMP then its entire boundary-scan register would be in the scan path with the FAC registers of U2. If the FAC architecture required going through CAPTURE-DR or UPDATE-DR, then programming performance would be affected and throughput would not be much better than using EXTEST alone. FAC data registers capture and update data in the UPDATE1/CAPTURE1state of the FAC TAP. Because of this U1 is does not go through UPDATE-DR or CAPTURE-DR during the programming operation of the FAC, thus giving FAC based programming independent of scan-chain length.
The FAC TAP also enables shifting of data on TDO during the APG2 state. During the APG2 state, status of the RDY/DQ7 bit indicating programming has completed, can be passed through the TDO back to the IEEE 1149.1 controller. When RDY is asserted this tells the 1149.1 controller (which is also FAC aware) to proceed from waiting in the APG2 state to move to the APG1 state to shift more data. By bringing the RDY bit through this way, optimal 1149.1 programming can be achieved without scanning a register or without having direct physical access to the RDY pin. The RDY status and the verification bit comparing the data read with the data written can be combined and returned on TDO during APTG2 as well. This can signal to the 1149.1 controller during the write process that the programming failed and further programming can be halted. Alleviating the need to have direct access to the RDY pin simplifies multi-PCB system level designs and PCBs with daughter cards enabling non-volatile memories to be programmed anywhere in-system with access to just the 5 wire 1149.1 bus.

FAC Performance

Comparing the programming times for the EXTEST Boundary Scan method, with that of the FAC, will show the improved performance of the FAC. Using an Intel StrataFLASH (28F128J3Ax16) as an example of a typical FLASH device, the following can be used to calculate the respective programming times:

- 128Mbits of memory, with a 16 bit data bus and 8M addresses.
- The typical burn time is 218us per buffer (16 words),
- 512k (524,288) buffer writes are required to program the entire 128Mbit FLASH.
- The PCB containing the FLASH and IC with Boundary Scan requires 736 BSR cells in the 1149.1 chain in order to access the FLASH device.
- The TCK clock rate is 10MHz.
- It takes 19 scan operations to program 1 buffer of 16 words.

Note that these specifications are taken from a demonstration PCB that includes the Intel FLASH device, and several other Boundary Scan devices. The FAC was programmed into an FPGA with access to the Intel Strataflash for the following calculations.

Using the Boundary Scan method, the time to program 1 buffer of 16 words is as follows:

Time for 1 buffer = 1/TCK * #BSR cells * #scans
= 0.1us * 736 * 19 = 1.4ms

To program the entire FLASH requires:

Time for FLASH = buffer time * #buffers
= 1.4ms * 524,288 = 734 seconds

Using the EXTEST method, it takes 12.2 minutes to program the entire FLASH. Note that this does not take into account the device burn time, which can be up to 114 seconds or more for the entire FLASH. This time is dominated by the shifting required to program the FLASH. As Boundary Scan chains on ICs and PCBs will continue to increase in length, the EXTEST method is not a practical solution.

In the FAC implementation for the StrataFLASH, there is a DATA DR that holds one write buffer worth of data, so the scan length is 16 data bits * 16 words, or 256 bits long. In addition there are two extra TCK cycles per buffer write, to account for the transition of the FAC TAP through *APG2-DR* and *Update2/Capture2-DR*. Using the FAC method, the time to program 1 buffer of 16 words is:

Time for 1 buffer = 1/TCK * #BSR cells * #scans
= 0.1us * 258 * 1 = 25.8us

To shift data to program the entire FLASH requires:

Time for FLASH = buffer time * #buffers
= 25.8us * 524,288 = 13.5 seconds

Since the shifting of the next data and the 'burn time' of current data can occur concurrently, the shifting time is negligible and the 'burn' time dominates. The Intel specification for this part is 218us typical per buffer with 524,288 buffers, the total typical burn time is approximately 114 seconds. However, in practice, the 'burn' time was significantly lower. The total programming time of some sample parts was approximately 104 seconds accounting for some software overhead. Most of the time is due to waiting for the FLASH, and is not because of shifting data, as was the case for EXTEST based programming. While verification can be done while programming with the FAC, data uniqueness was required for the customer and the FLASH was read back as a separate function. This takes 18 TCK cycles per word for 8M locations, which at 10Mhz takes approximately 15 seconds, 4 seconds at a 40Mhz TCK rate.

Related work

After the invention of the FAC, but prior to the granting of the US patent, a method in the spirit of the FAC technique has been presented at ITC [4]. The reader should not assume the technique described in the ITC paper is in the public domain. It should be noted from the above descriptions that a full FAC implementation is more generic than what was described. In [4], the method requires going through CAPTURE-DR and UPDATE-DR after shifting each address and data. The programming times given are good only for a single device in the scan-chain. If the device were in the scan path with other on-board devices then the address and data would have to be shifted through those devices for each word to program. On PCBs with many devices in EXTEST, the programming times would be impacted and as long as EXTEST programming times. The same problem exists for performing a scan operation to read the DQ7 or RDY bits. If the device is in a scan-chain with other devices, it can take far longer to shift out the status bit than it takes for the typical 'burn time'. The verification immediately after programming also described is a FLASH memory only solution, as it would not be appropriate for EEPROM programming or SRAM test. No mention in the paper was made on how the pass/fail status could be examined during programming so programming would halt early on for failing devices. Further, the verification method described in [4] is for production only, it would not allow data to be downloaded from a memory to a PC for the purpose of debugging during bring-up. The ITC paper does not describe whether it can be sequenced by a system clock or by TCK. Without access to the system clock, at-speed tests are not possible. Further, if a state machine is sequenced only by the TCK, then provision must be provided for handling sequences that are longer than the amount of data to shift. For instance, writing to a Serial EEPROM as can be done with the FAC requires many TCK cycles which can be handled by the FAC in the APG2 state.

Conclusions and future work

The FAC provides a novel solution for programming external FLASH in a production environment. It avoids many of the problems with traditional FLASH programming methods and boundary-scan based programming while it lowers overall product costs and downstream PCB manufacturing costs. The programming performance of the FAC has been shown to be superior to other in-system programming methods, programming FLASH memories as fast as off-board or direct access programming solutions can. Similar performance can be achieved for other non-volatile memories such as I2C and SPI EEPROMS, NAND FLASH. The flexibility of the FAC allows it to be implemented in an ASIC or CPU where the target PCB scan-chain length is unknown and the non-volatile memory type is unknown at the time of design.

The FAC enables an IEEE Std. 1149.1 bus to be used as a central bus for in-system configuration of all on-board non-volatile devices, and it provides a single non-volatile programming method that can be used on systems in the field, during prototype bring-up, or for manufacturing production. This enables last minute updates to non-volatile to be done during manufacturing, and it eliminates the need for costly inventories of pre-programmed devices or costly capital equipment to program in-line. In addition, manufacturing can realize shorter programming times and improved throughput on the manufacturing floor.

Boundary-Scan test techniques introduce the similar problems for DDDRAM and SDRAM test and debug. The long scan-chains and typical slow speed make it impossible to test the interconnects to most dynamic RAM. Since the density of the typical dynamic memory is high, small vias (or no vias) must be used during the design of the PCB. Systems integrators have difficulty in inserting additional test pads for testing these components with direct in-circuit techniques. Therefore, significant time in design by the System integrator and savings in test development can be achieved if the silicon designer can envision the environment his silicon will be used by his customer. Since the FAC can operate at the system clock rate, the FAC can perform at-speed interconnect testing to these memories without requiring the system's integrator to insert test pads. The FAC mechanism facilitates enables postmortem memory dumps of these external memories without disturbing or requiring the system CPU to function or execute code. This has two benefits in that it can be used by the silicon designer during prototype validation and then later by the customer during system bring-up. Table 1, shows how a single FAC access controller can be used to support EEPROM, FLASH, NAND, DRAM and other memory types that may be used. Since the IC/SoC design may not be able to predict what external memory type the system integrator may used, the configuration of the fast access controller is done at run time through standard IEEE 1149.1 software.

References

[1] IEEE Std 1149.1b-1994, "IEEE Standard Test Access Port and Boundary-Scan Architecture", Institute of Electrical and Electronic Engineers, Inc., New York, NY, USA.

[2] Ricchetti, M., Clark, CJ, Dervisoglu, B., "Method and Apparatus for Providing Optimized Access to Circuits for Debug, Programming, and Test", US Patent No. US 6,594,802, US Patent and Trademark Office, Washington, D.C., March 23, 2000. http://www.uspto.gov

[3] Ricchetti, M. Clark, CJ, Dervisoglu, B., "Method and Apparatus for Providing Optimized Access to Circuits for Debug, Programming, and Test", PCT Patent Application WO0171876, World Intellectual Property Organization, Geneva, Switzerland, March 23, 2000.

[4] de Jong, F., Biewenga , A.S., van Geest, D.C.L., Waayers, T.F., "Testing and Programming FLASH Memories on Assemblies During High Volume Production", Proceedings of the IEEE International Test Conference 2001, pp. 470-479.

[5] CJ Clark, Mike Ricchetti, "Infrastructure IP for Configuration and Test of Boards and Systems", IEEE Design & Test of Computers, vol. 20, no. 3, May-June 2003, pp. 78-87.

[6] Ricchetti, M. Clark, CJ, "Method and Apparatus for Optimized Parallel Testing and Access of Electronic Circuits", *US Patent Application 2003009715*, US Patent and Trademark Office, Washington, D.C., July 5, 2001. http://www.uspto.gov.

[7] Clark, CJ, Ricchetti, M., "Method and Apparatus for Optimized Parallel Testing and Access of Electronic Circuits", *PCT Patent Application WO03005050*, World Intellectual Property Organization, Geneva, Switzerland, July 5, 2001. http://ep.espacenet.com