US 20100223453A1

(19) United States
(12) Patent Application Publication
WALMSLEY
(10) Pub. No.: US 2010/0223453 A1
(43) Pub. Date: Sep. 2, 2010

(54) INTEGRATED CIRCUIT FOR VALIDATING AND DECRYPTING SOFTWARE DATA

(75) Inventor: **Simon Robert WALMSLEY**, Balmain (AU)

Correspondence Address:
SILVERBROOK RESEARCH PTY LTD
393 DARLING STREET
BALMAIN 2041 (AU)

(73) Assignee: **Silverbrook Research Pty Ltd**

(21) Appl. No.: **12/778,966**

(22) Filed: **May 12, 2010**

Related U.S. Application Data

(60) Continuation of application No. 10/754,536, filed on Jan. 12, 2004, which is a division of application No. 10/727,251, filed on Dec. 2, 2003, now Pat. No. 7,188, 282.

(30) **Foreign Application Priority Data**

Dec. 2, 2002	(AU)	2002953134
Dec. 2, 2002	(AU)	2002953135

Publication Classification

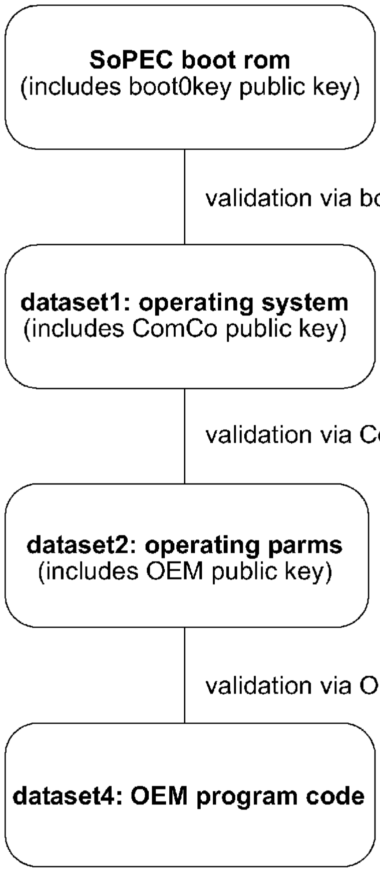
(51) **Int. Cl.**

G06F 21/22	(2006.01)
G06F 9/24	(2006.01)

(52) **U.S. Cl.** **713/2**; 713/190

(57) **ABSTRACT**

An integrated circuit is provided. The IC runs a boot program that verifies programs before the programs can be loaded onto, or run by, the IC by verifying whether the programs are signed with a boot key, verifies, with the boot program, a developmental boot program signed with the boot key which verifies developmental programs before the developmental programs can be loaded onto, or run by, the IC by verifying whether the IC has a predetermined identifier, and loads the verified developmental boot program and run the loaded developmental booth program thereby enabling loading or running of the developmental programs on the IC if the IC has the predetermined identifier. The IC is programmed with program code configured to receive encrypted software data, decrypt the software data, and validate the software data. The decrypted software is executed only when the validation is successful.


SoPEC boot rom
(includes boot0key public key)
validation via boot0key
dataset1: operating system
(includes ComCo public key)
validation via ComCo key
dataset2: operating parms
(includes OEM public key)
validation via OEM key
dataset4: OEM program code

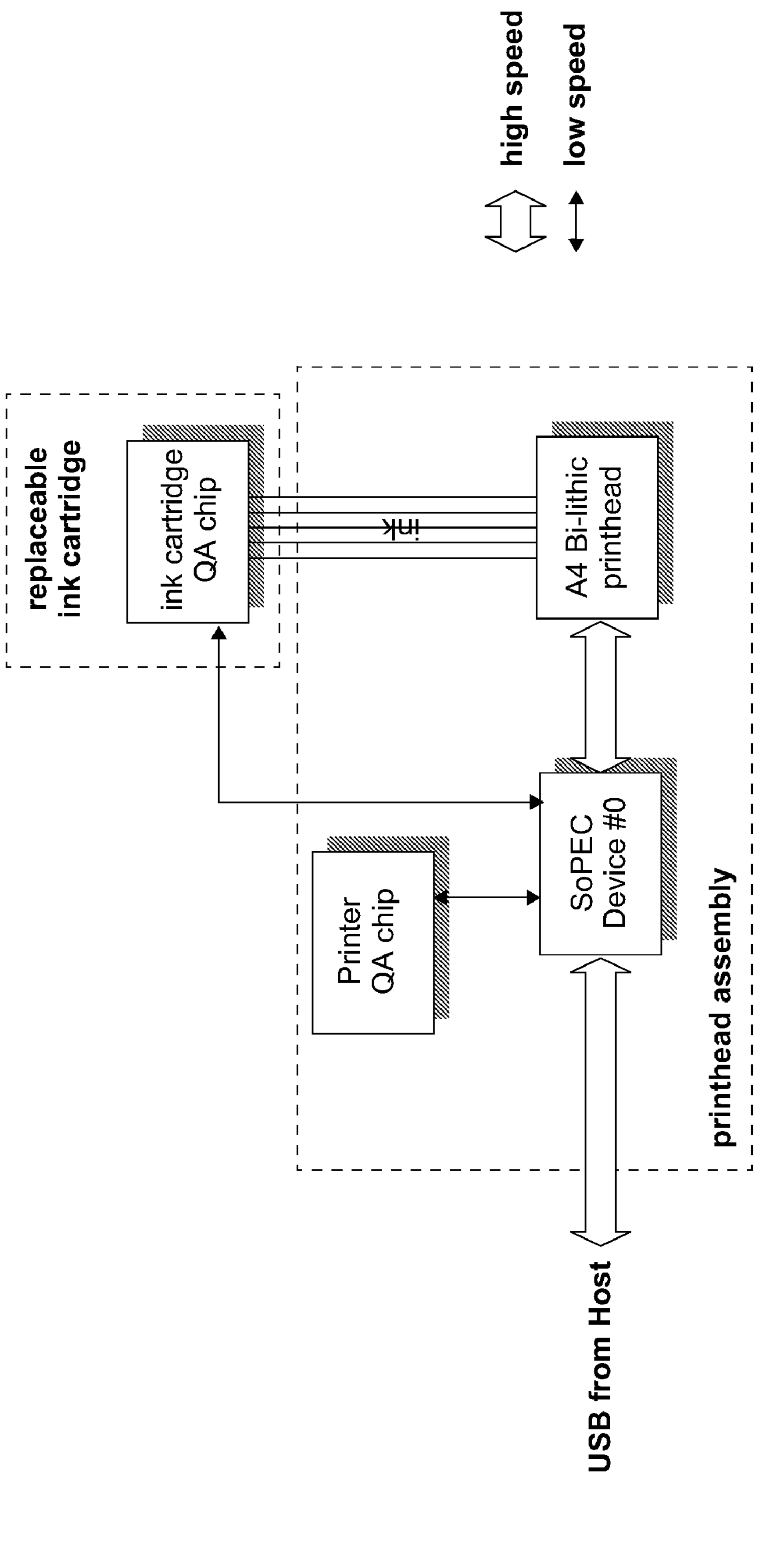

replaceable ink cartridge
ink cartridge QA chip
ink
A4 Bi-lithic printhead
SoPEC Device #0
Printer QA chip
printhead assembly
USB from Host
high speed
low speed

FIG. 1

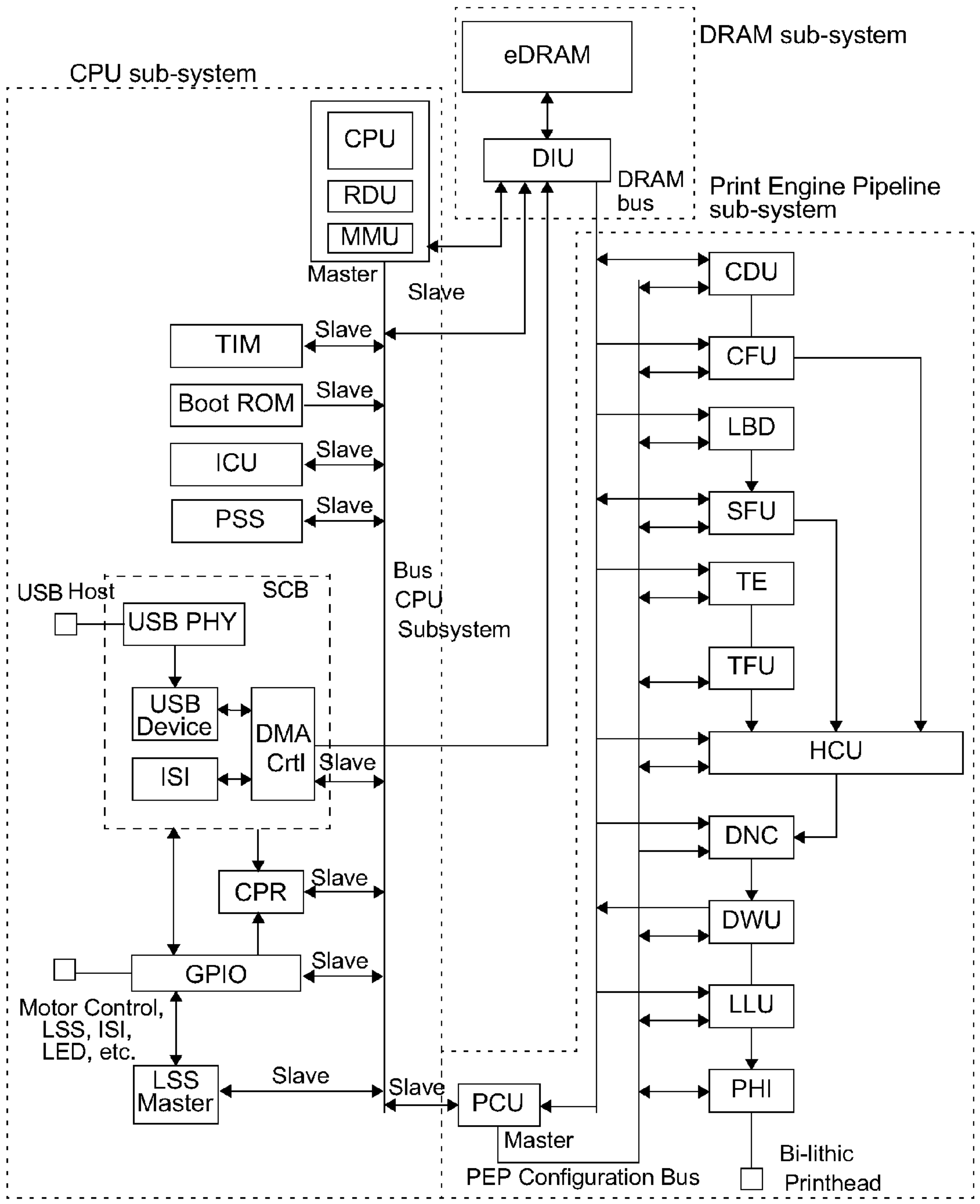

DRAM sub-system
eDRAM
DIU
DRAM bus
CPU sub-system
CPU
RDU
MMU
Master
Slave
Print Engine Pipeline sub-system
TIM
Slave
Boot ROM
Slave
ICU
Slave
PSS
Slave
CDU
CFU
LBD
SFU
TE
TFU
HCU
DNC
DWU
LLU
PHI
Bus CPU Subsystem
USB Host
SCB
USB PHY
USB Device
DMA Crtl
ISI
Slave
CPR
Slave
GPIO
Slave
Motor Control, LSS, ISI, LED, etc.
LSS Master
Slave
Slave
PCU
Master
PEP Configuration Bus
Bi-lithic Printhead

FIG. 2

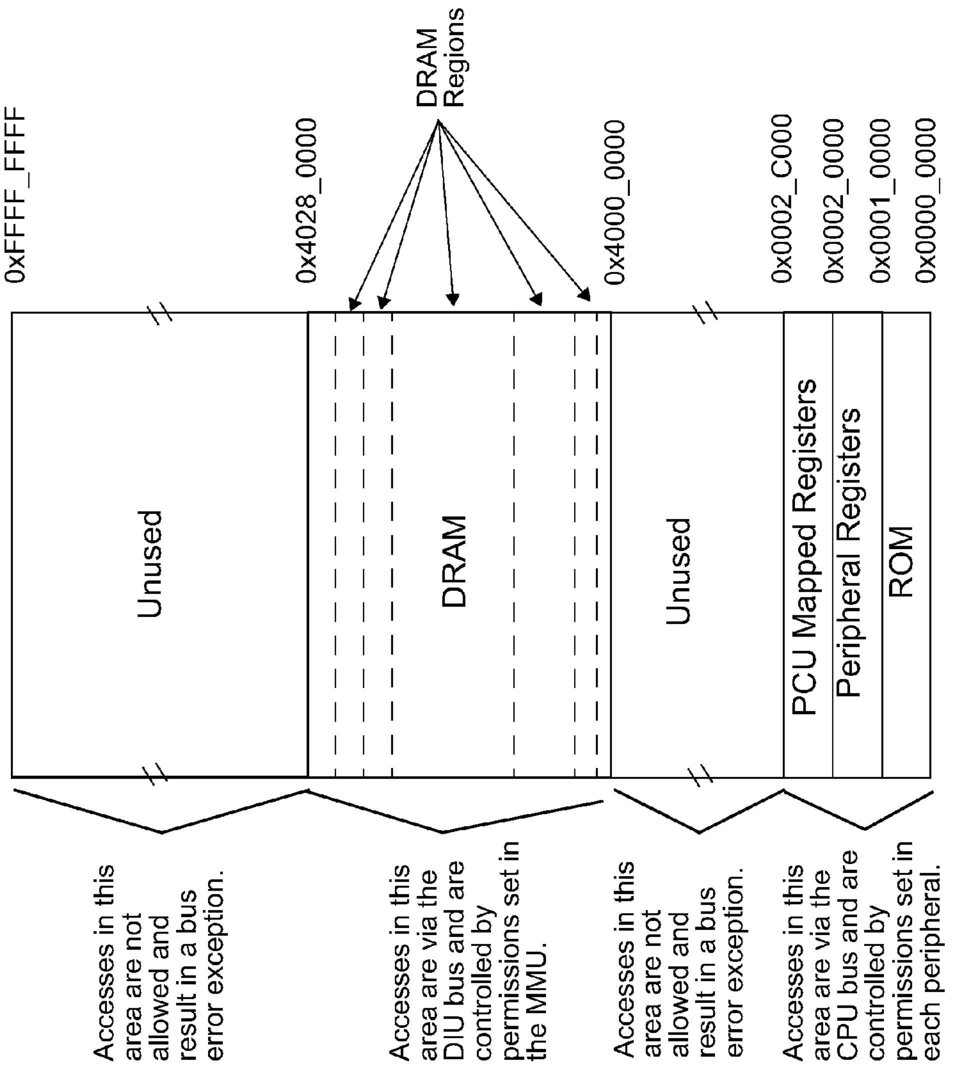

0xFFFF_FFFF
0x4028_0000
DRAM Regions
0x4000_0000
0x0002_C000
0x0002_0000
0x0001_0000
0x0000_0000
Unused
DRAM
Unused
PCU Mapped Registers
Peripheral Registers
ROM
Accesses in this area are not allowed and result in a bus error exception.
Accesses in this area are via the DIU bus and are controlled by permissions set in the MMU.
Accesses in this area are not allowed and result in a bus error exception.
Accesses in this area are via the CPU bus and are controlled by permissions set in each peripheral.

FIG. 3

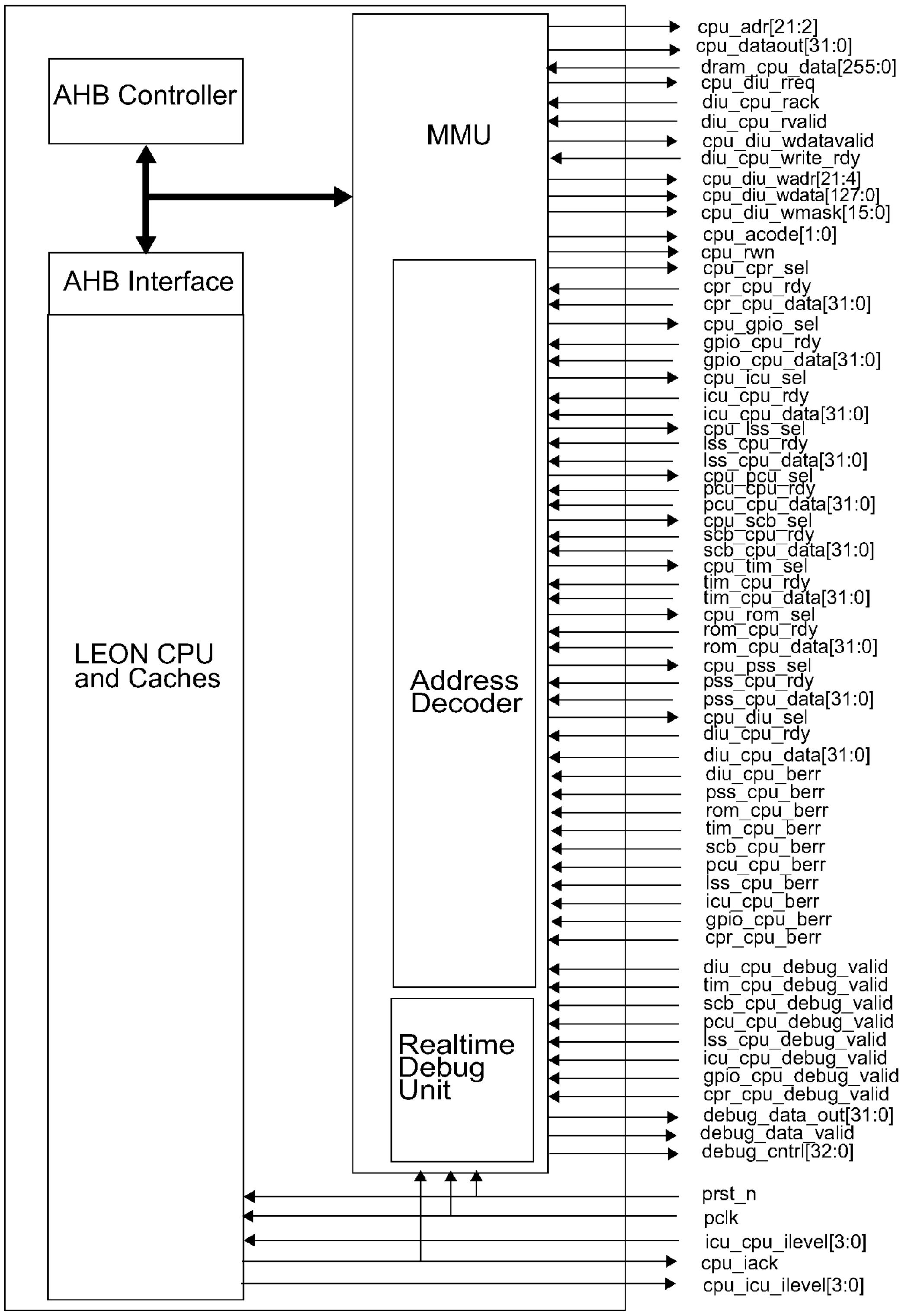

AHB Controller
AHB Interface
LEON CPU and Caches
MMU
Address Decoder
Realtime Debug Unit
cpu_adr[21:2]
cpu_dataout[31:0]
dram_cpu_data[255:0]
cpu_diu_rreq
diu_cpu_rack
diu_cpu_rvalid
cpu_diu_wdatavalid
diu_cpu_write_rdy
cpu_diu_wadr[21:4]
cpu_diu_wdata[127:0]
cpu_diu_wmask[15:0]
cpu_acode[1:0]
cpu_rwn
cpu_cpr_sel
cpr_cpu_rdy
cpr_cpu_data[31:0]
cpu_gpio_sel
gpio_cpu_rdy
gpio_cpu_data[31:0]
cpu_icu_sel
icu_cpu_rdy
icu_cpu_data[31:0]
cpu_lss_sel
lss_cpu_rdy
lss_cpu_data[31:0]
cpu_pcu_sel
pcu_cpu_rdy
pcu_cpu_data[31:0]
cpu_scb_sel
scb_cpu_rdy
scb_cpu_data[31:0]
cpu_tim_sel
tim_cpu_rdy
tim_cpu_data[31:0]
cpu_rom_sel
rom_cpu_rdy
rom_cpu_data[31:0]
cpu_pss_sel
pss_cpu_rdy
pss_cpu_data[31:0]
cpu_diu_sel
diu_cpu_rdy
diu_cpu_data[31:0]
diu_cpu_berr
pss_cpu_berr
rom_cpu_berr
tim_cpu_berr
scb_cpu_berr
pcu_cpu_berr
lss_cpu_berr
icu_cpu_berr
gpio_cpu_berr
cpr_cpu_berr
diu_cpu_debug_valid
tim_cpu_debug_valid
scb_cpu_debug_valid
pcu_cpu_debug_valid
lss_cpu_debug_valid
icu_cpu_debug_valid
gpio_cpu_debug_valid
cpr_cpu_debug_valid
debug_data_out[31:0]
debug_data_valid
debug_cntrl[32:0]
prst_n
pclk
icu_cpu_ilevel[3:0]
cpu_iack
cpu_icu_ilevel[3:0]

FIG. 4

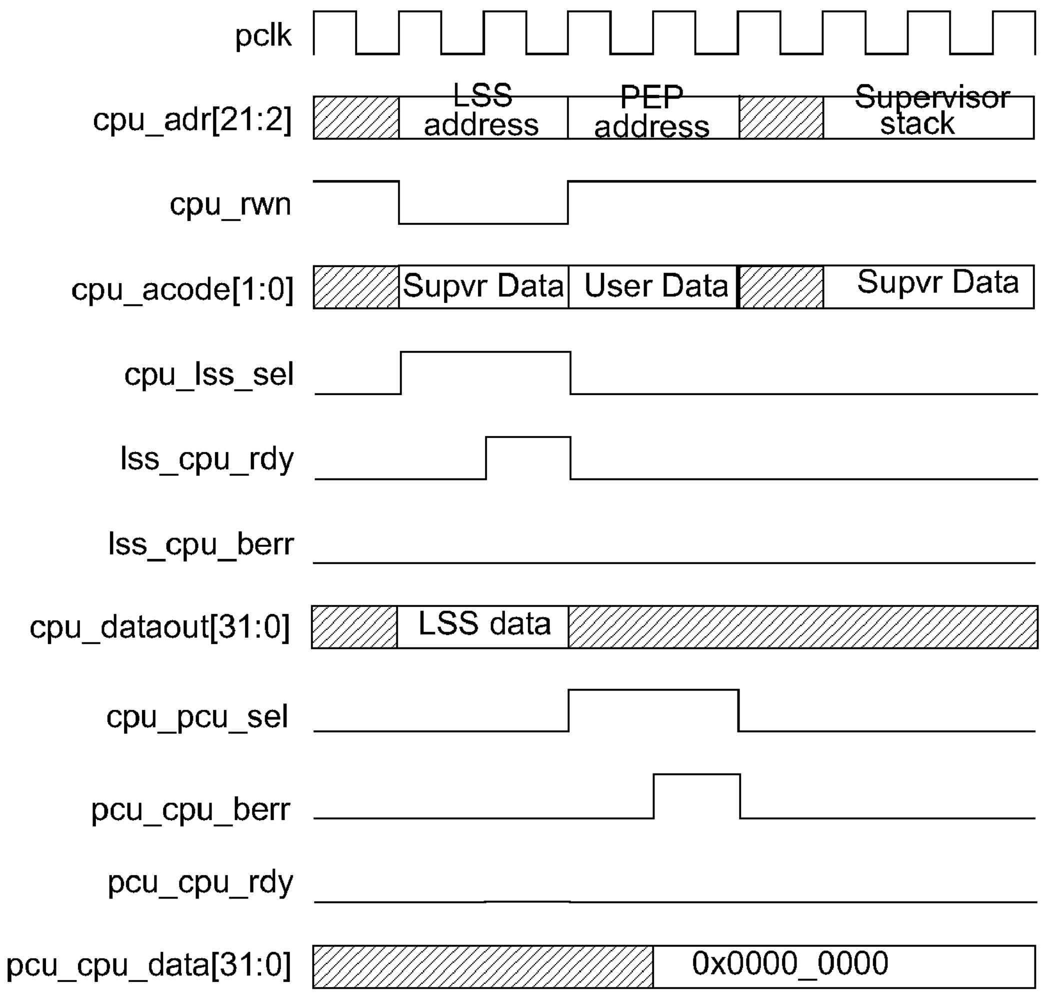

pclk
cpu_adr[21:2]
LSS address
PEP address
Supervisor stack
cpu_rwn
cpu_acode[1:0]
Supvr Data
User Data
Supvr Data
cpu_lss_sel
lss_cpu_rdy
lss_cpu_berr
cpu_dataout[31:0]
LSS data
cpu_pcu_sel
pcu_cpu_berr
pcu_cpu_rdy
pcu_cpu_data[31:0]
0x0000_0000

FIG. 5

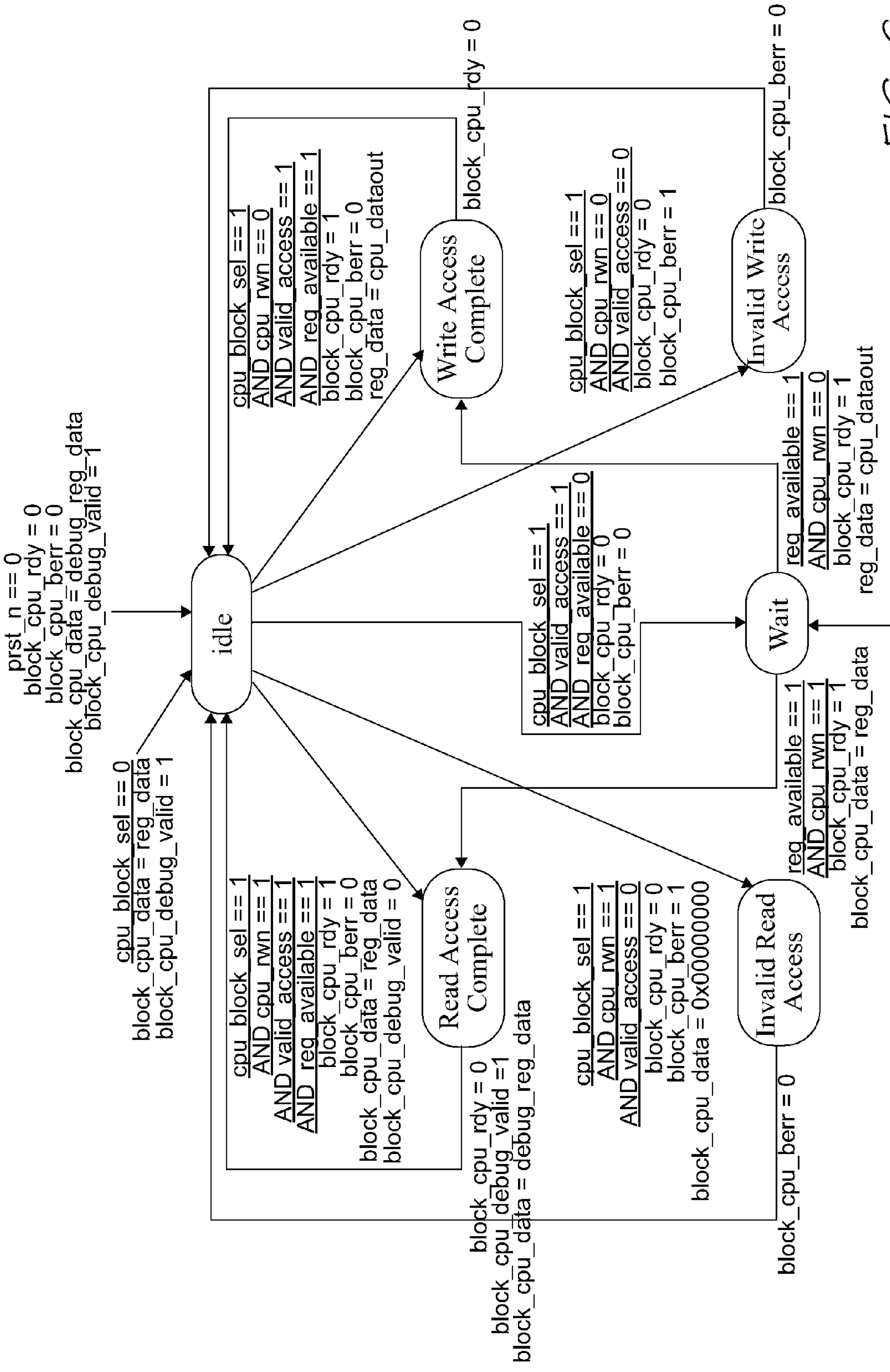

prst_n == 0
block_cpu_rdy = 0
block_cpu_berr = 0
block_cpu_data = debug_reg_data
block_cpu_debug_valid = 1
idle
cpu_block_sel == 0
block_cpu_data = reg_data
block_cpu_debug_valid = 1
cpu_block_sel == 1
AND cpu_rwn == 0
AND valid_access == 1
AND reg_available == 1
block_cpu_rdy = 1
block_cpu_berr = 0
reg_data = cpu_dataout
Write Access Complete
block_cpu_rdy = 0
cpu_block_sel == 1
AND cpu_rwn == 0
AND valid_access == 0
block_cpu_rdy = 0
block_cpu_berr = 1
Invalid Write Access
block_cpu_berr = 0
cpu_block_sel == 1
AND valid_access == 1
AND reg_available == 0
block_cpu_rdy = 0
block_cpu_berr = 0
Wait
reg_available == 1
AND cpu_rwn == 0
block_cpu_rdy = 1
reg_data = cpu_dataout
reg_available == 0
reg_available == 1
AND cpu_rwn == 1
block_cpu_rdy = 1
block_cpu_data = reg_data
cpu_block_sel == 1
AND cpu_rwn == 1
AND valid_access == 1
AND reg_available == 1
block_cpu_rdy = 1
block_cpu_berr = 0
block_cpu_data = reg_data
block_cpu_debug_valid = 0
Read Access Complete
block_cpu_rdy = 0
block_cpu_debug_valid = 1
block_cpu_data = debug_reg_data
cpu_block_sel == 1
AND cpu_rwn == 1
AND valid_access == 0
block_cpu_rdy = 0
block_cpu_berr = 1
block_cpu_data = 0x00000000
Invalid Read Access
block_cpu_berr = 0

FIG. 6

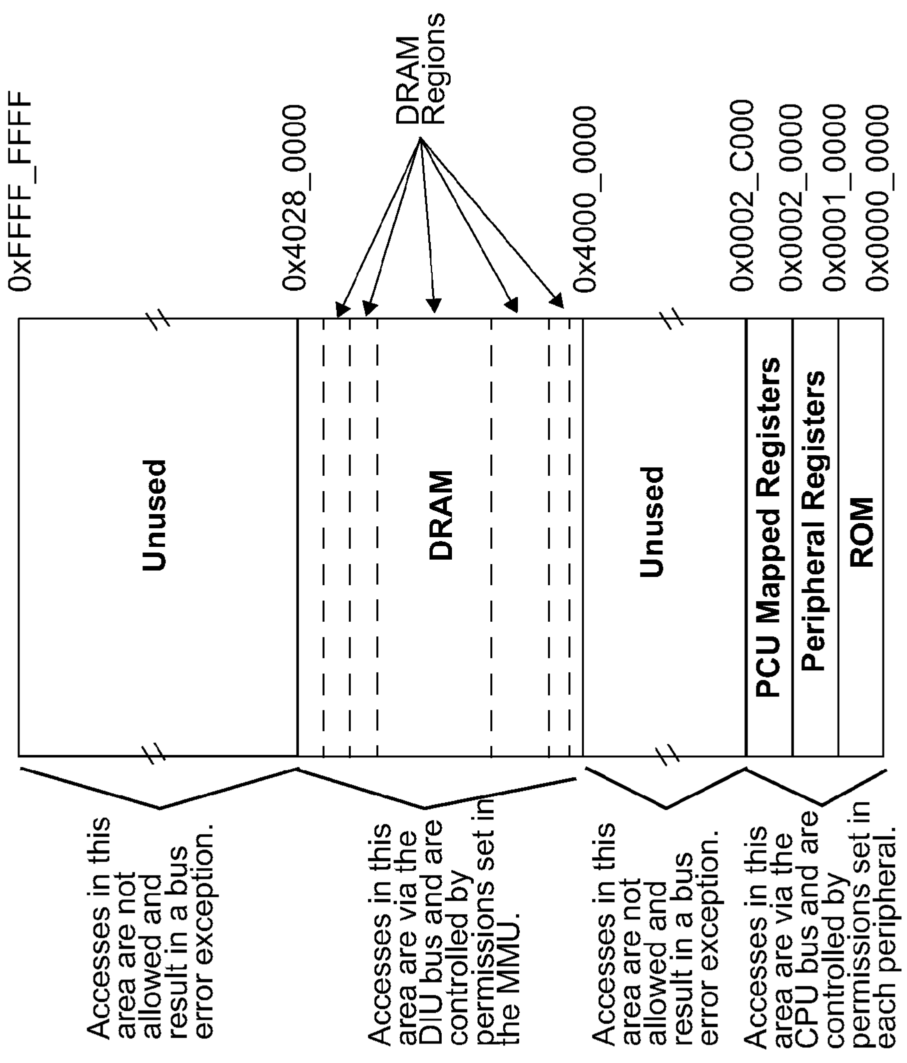

0xFFFF_FFFF
0x4028_0000
DRAM Regions
0x4000_0000
0x0002_C000
0x0002_0000
0x0001_0000
0x0000_0000
Unused
DRAM
Unused
PCU Mapped Registers
Peripheral Registers
ROM
Accesses in this area are not allowed and result in a bus error exception.
Accesses in this area are via the DIU bus and are controlled by permissions set in the MMU.
Accesses in this area are not allowed and result in a bus error exception.
Accesses in this area are via the CPU bus and are controlled by permissions set in each peripheral.

FIG. 7

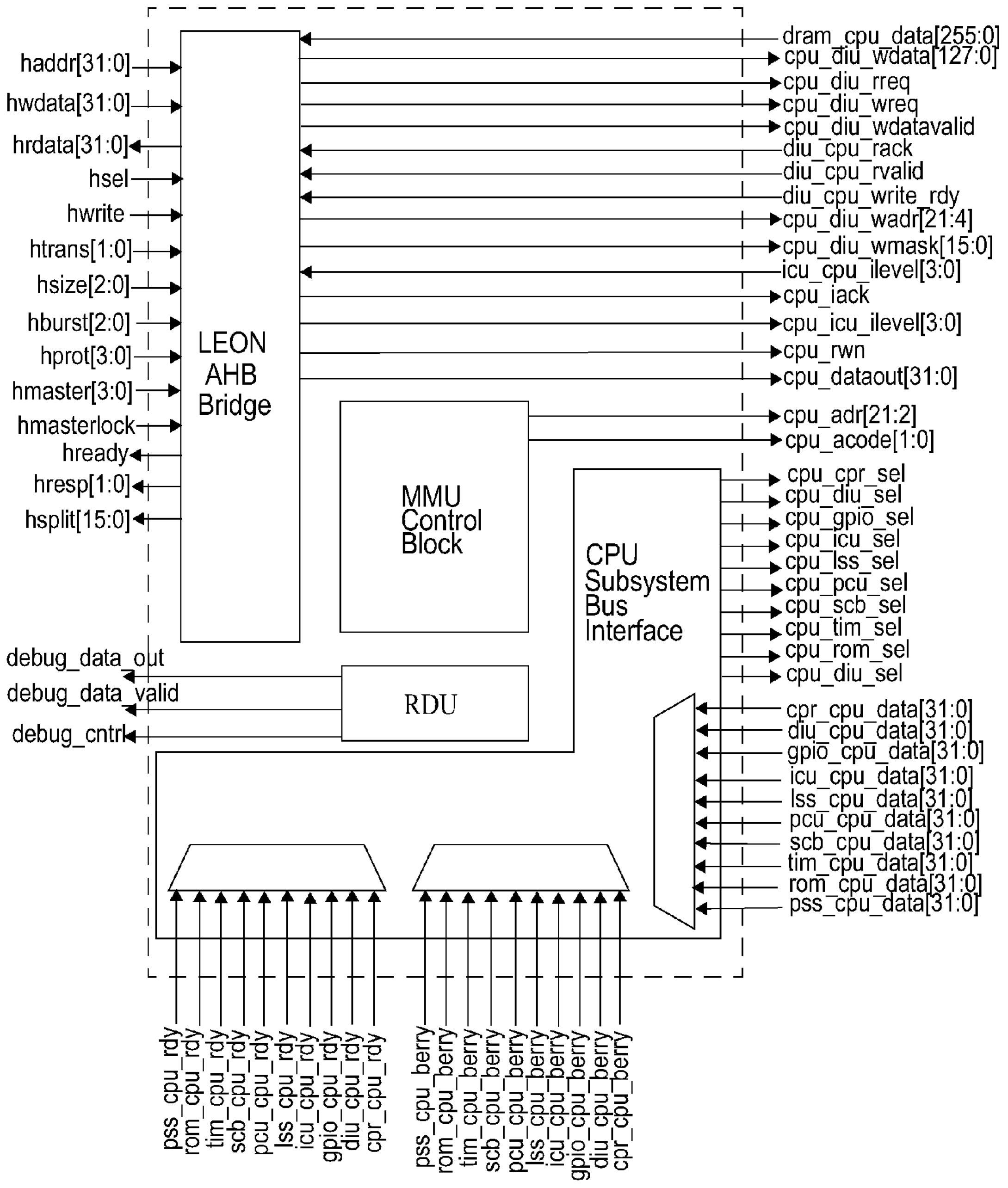

haddr[31:0]
hwdata[31:0]
hrdata[31:0]
hsel
hwrite
htrans[1:0]
hsize[2:0]
hburst[2:0]
hprot[3:0]
hmaster[3:0]
hmasterlock
hready
hresp[1:0]
hsplit[15:0]
LEON AHB Bridge
MMU Control Block
CPU Subsystem Bus Interface
RDU
debug_data_out
debug_data_valid
debug_cntrl
dram_cpu_data[255:0]
cpu_diu_wdata[127:0]
cpu_diu_rreq
cpu_diu_wreq
cpu_diu_wdatavalid
diu_cpu_rack
diu_cpu_rvalid
diu_cpu_write_rdy
cpu_diu_wadr[21:4]
cpu_diu_wmask[15:0]
icu_cpu_ilevel[3:0]
cpu_iack
cpu_icu_ilevel[3:0]
cpu_rwn
cpu_dataout[31:0]
cpu_adr[21:2]
cpu_acode[1:0]
cpu_cpr_sel
cpu_diu_sel
cpu_gpio_sel
cpu_icu_sel
cpu_lss_sel
cpu_pcu_sel
cpu_scb_sel
cpu_tim_sel
cpu_rom_sel
cpu_diu_sel
cpr_cpu_data[31:0]
diu_cpu_data[31:0]
gpio_cpu_data[31:0]
icu_cpu_data[31:0]
lss_cpu_data[31:0]
pcu_cpu_data[31:0]
scb_cpu_data[31:0]
tim_cpu_data[31:0]
rom_cpu_data[31:0]
pss_cpu_data[31:0]
pss_cpu_rdy
rom_cpu_rdy
tim_cpu_rdy
scb_cpu_rdy
pcu_cpu_rdy
lss_cpu_rdy
icu_cpu_rdy
gpio_cpu_rdy
diu_cpu_rdy
cpr_cpu_rdy
pss_cpu_berry
rom_cpu_berry
tim_cpu_berry
scb_cpu_berry
pcu_cpu_berry
lss_cpu_berry
icu_cpu_berry
gpio_cpu_berry
diu_cpu_berry
cpr_cpu_berry

FIG. 8

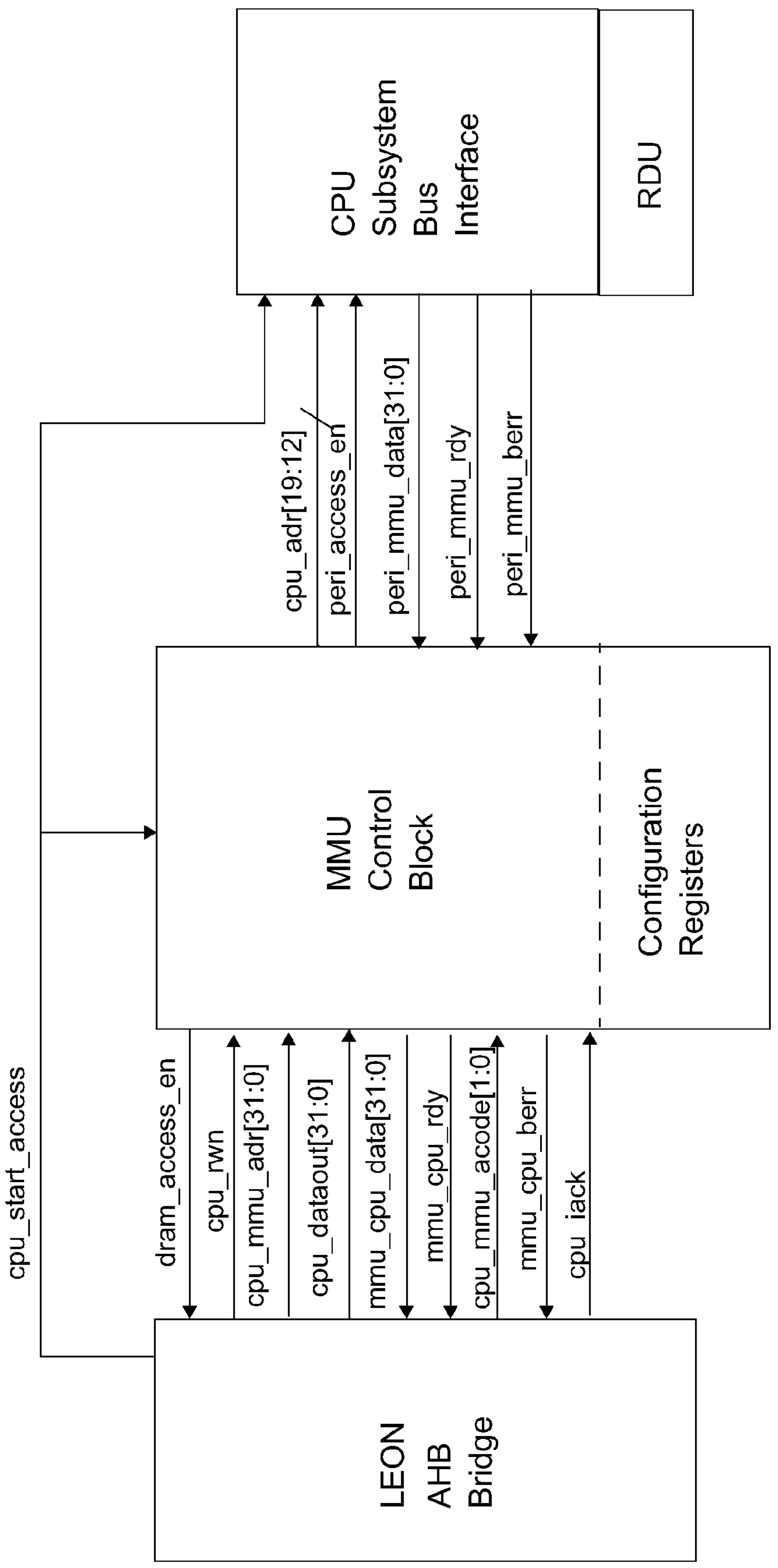

CPU Subsystem Bus Interface
RDU
MMU Control Block
Configuration Registers
LEON AHB Bridge
cpu_adr[19:12]
peri_access_en
peri_mmu_data[31:0]
peri_mmu_rdy
peri_mmu_berr
cpu_start_access
dram_access_en
cpu_rwn
cpu_mmu_adr[31:0]
cpu_dataout[31:0]
mmu_cpu_data[31:0]
mmu_cpu_rdy
cpu_mmu_acode[1:0]
mmu_cpu_berr
cpu_iack

FIG. 9

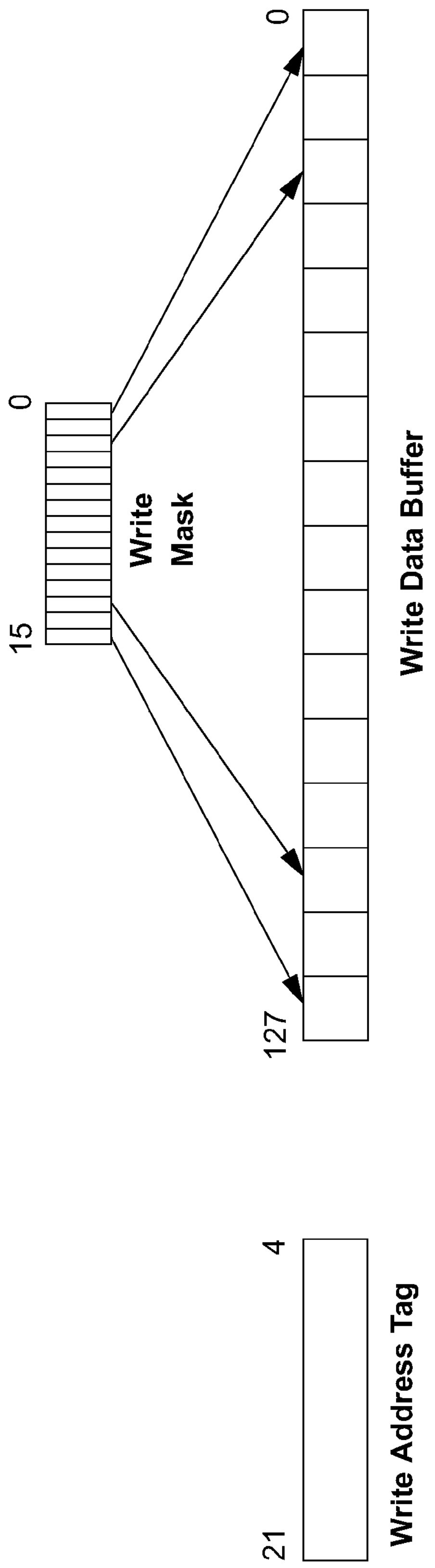


FIG. 10

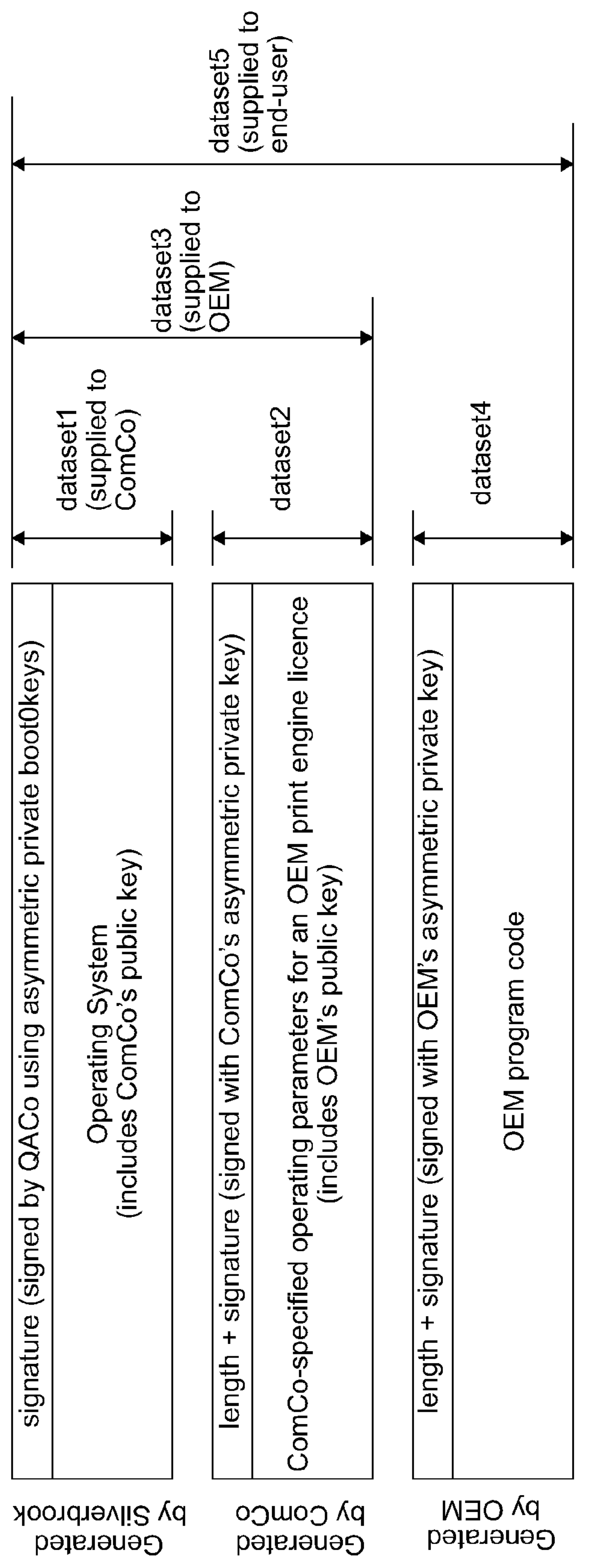

signature (signed by QACo using asymmetric private boot0keys)
Operating System (includes ComCo's public key)
Generated by Silverbrook
length + signature (signed with ComCo's asymmetric private key)
ComCo-specified operating parameters for an OEM print engine licence (includes OEM's public key)
Generated by ComCo
length + signature (signed with OEM's asymmetric private key)
OEM program code
Generated by OEM
dataset1 (supplied to ComCo)
dataset2
dataset4
dataset3 (supplied to OEM)
dataset5 (supplied to end-user)

FIG. 11

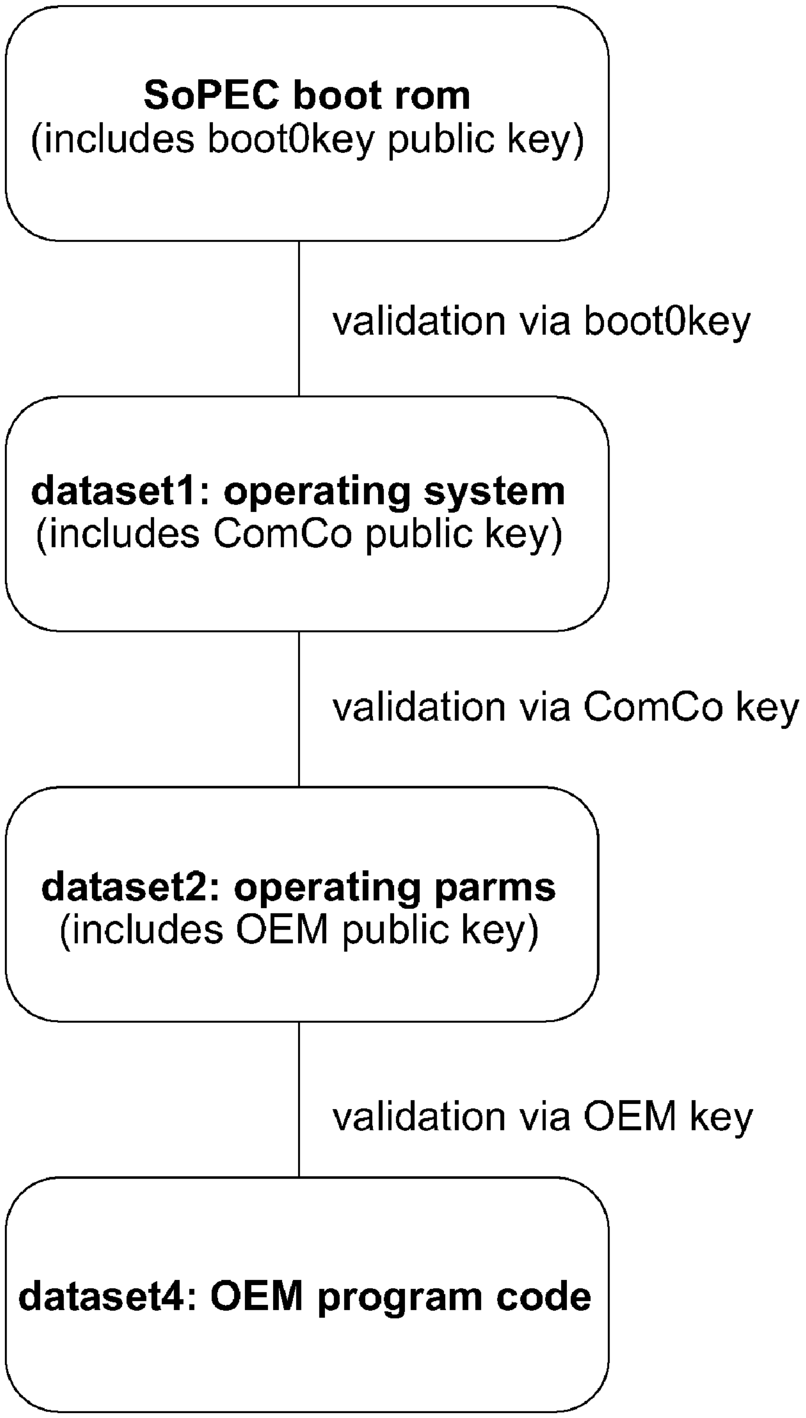

SoPEC boot rom
(includes boot0key public key)
validation via boot0key
dataset1: operating system
(includes ComCo public key)
validation via ComCo key
dataset2: operating parms
(includes OEM public key)
validation via OEM key
dataset4: OEM program code

FIG. 12

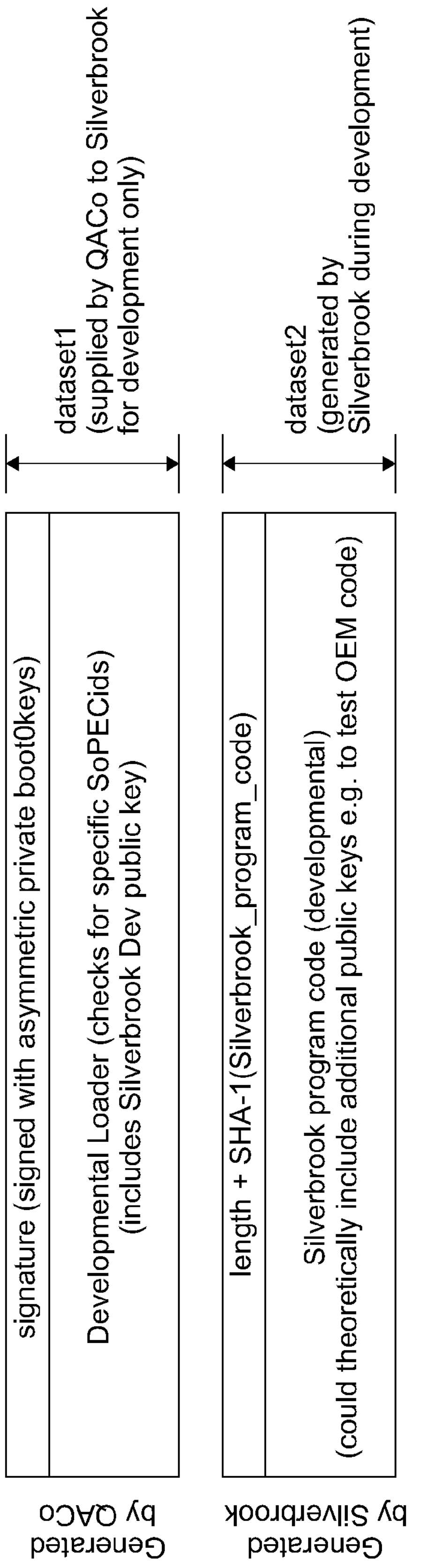

signature (signed with asymmetric private boot0keys)
Developmental Loader (checks for specific SoPECids)
(includes Silverbrook Dev public key)
dataset1
(supplied by QACo to Silverbrook
for development only)
Generated by QACo
length + SHA-1(Silverbrook_program_code)
Silverbrook program code (developmental)
(could theoretically include additional public keys e.g. to test OEM code)
dataset2
(generated by
Silverbrook during development)
Generated by Silverbrook

FIG. 13

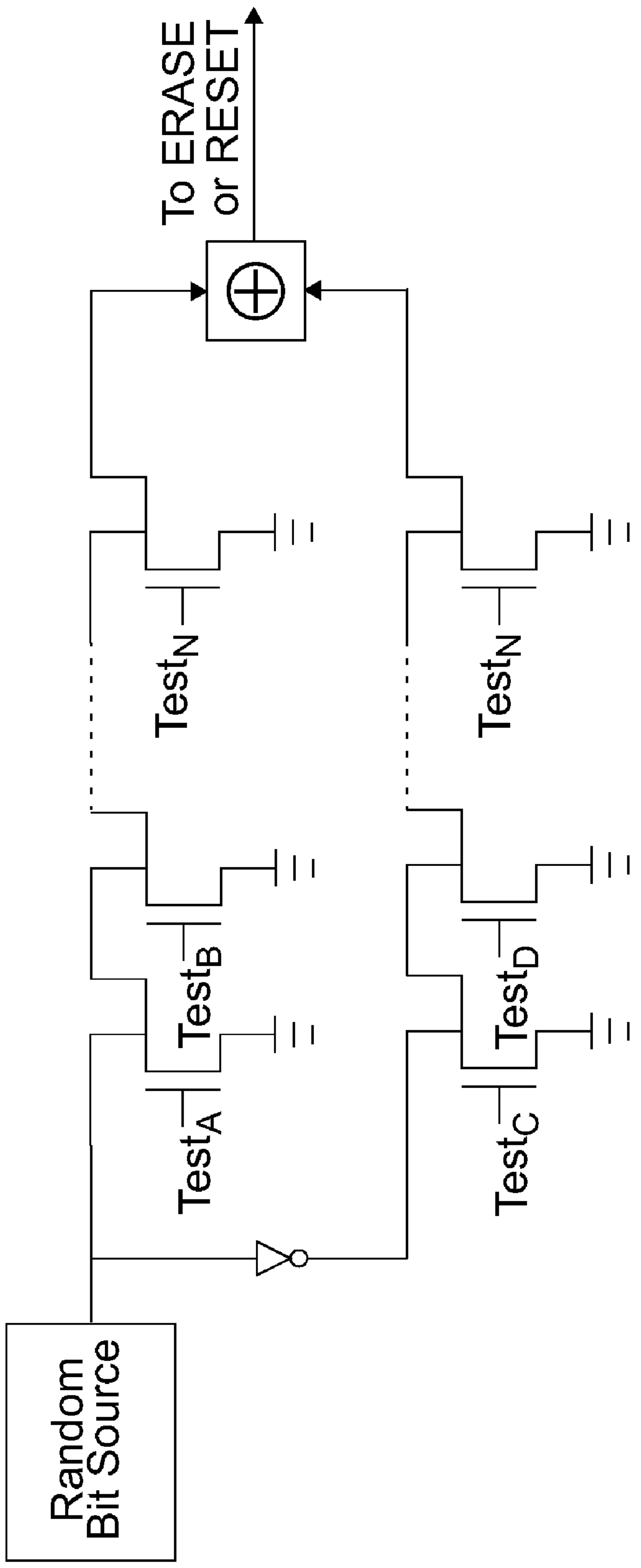

To ERASE or RESET
$\oplus$
Test$_N$
Test$_N$
Test$_B$
Test$_D$
Test$_A$
Test$_C$
Random Bit Source

FIG. 14

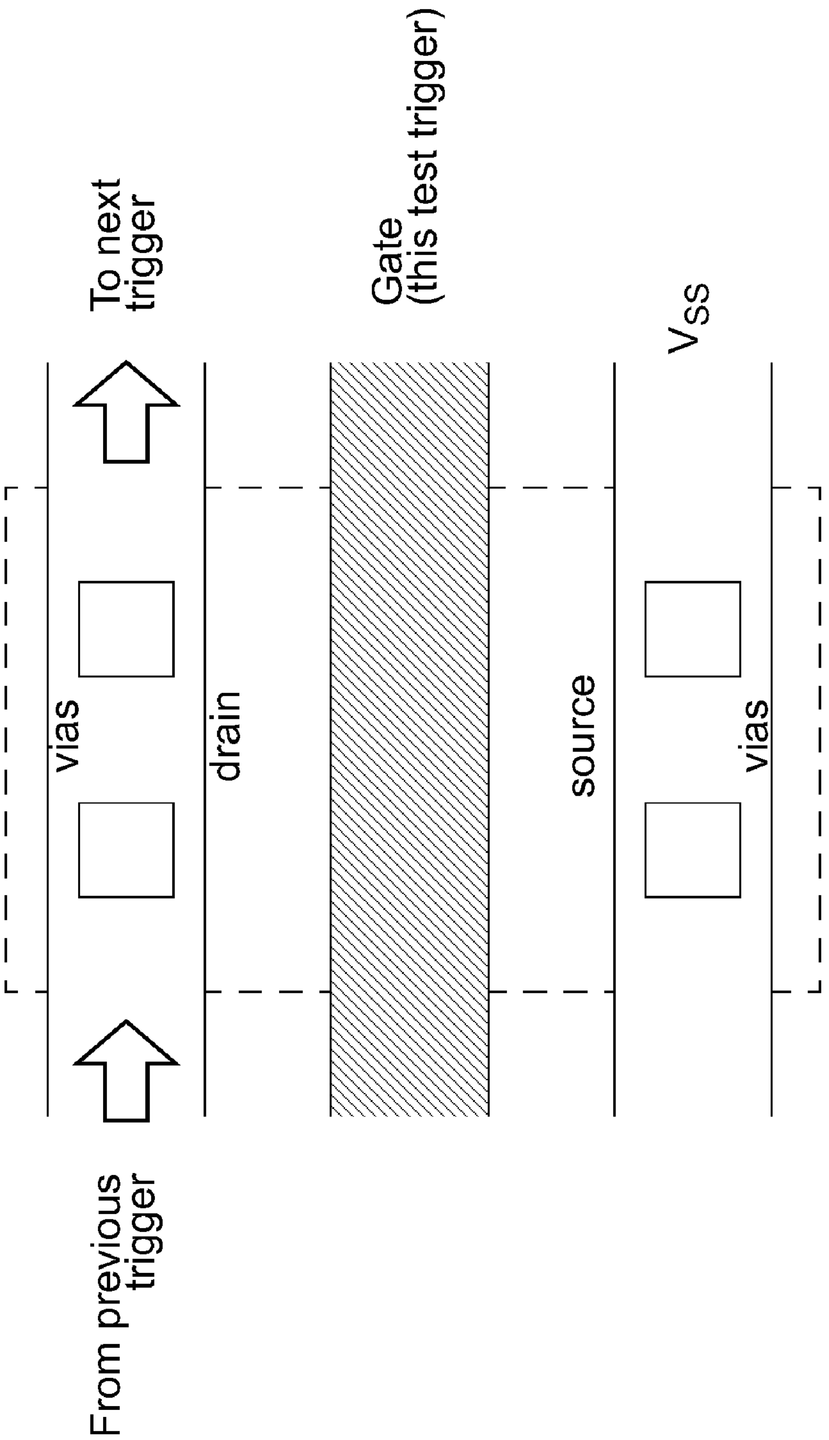

From previous trigger
To next trigger
vias
drain
Gate (this test trigger)
source
vias
V_{SS}

FIG. 15

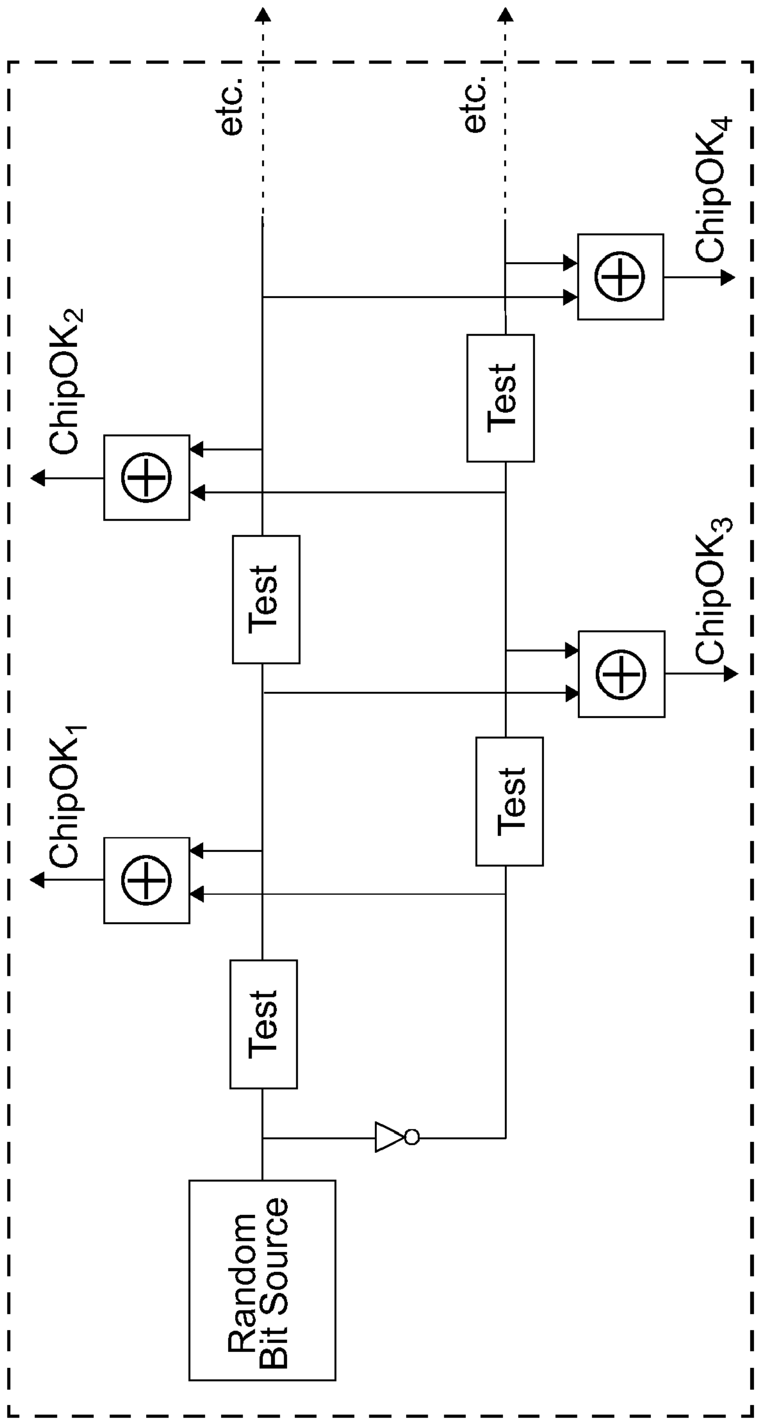

Random Bit Source
Test
Test
Test
Test
ChipOK$_1$
ChipOK$_2$
ChipOK$_3$
ChipOK$_4$
etc.
etc.

FIG. 16

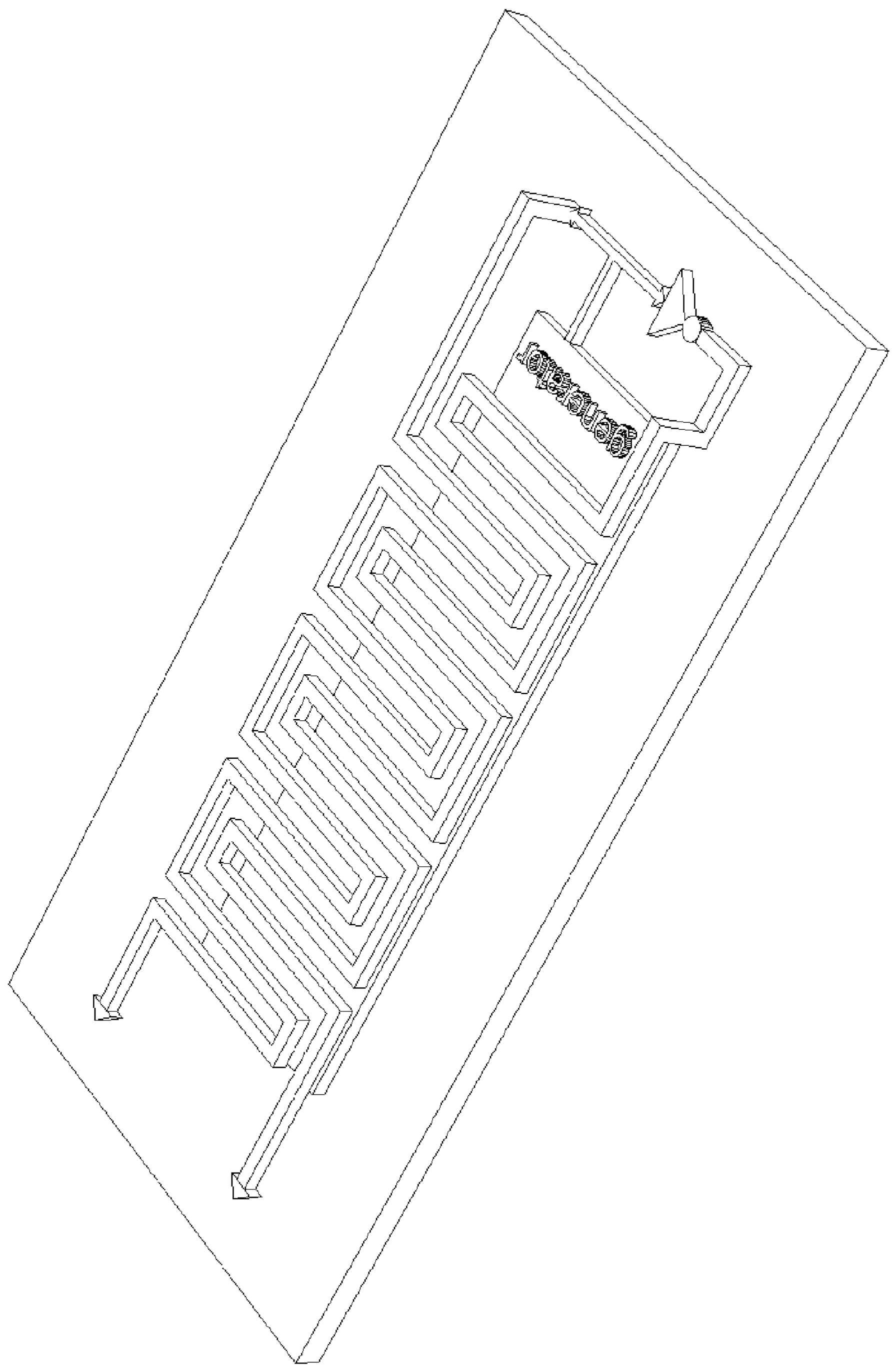


FIG. 17

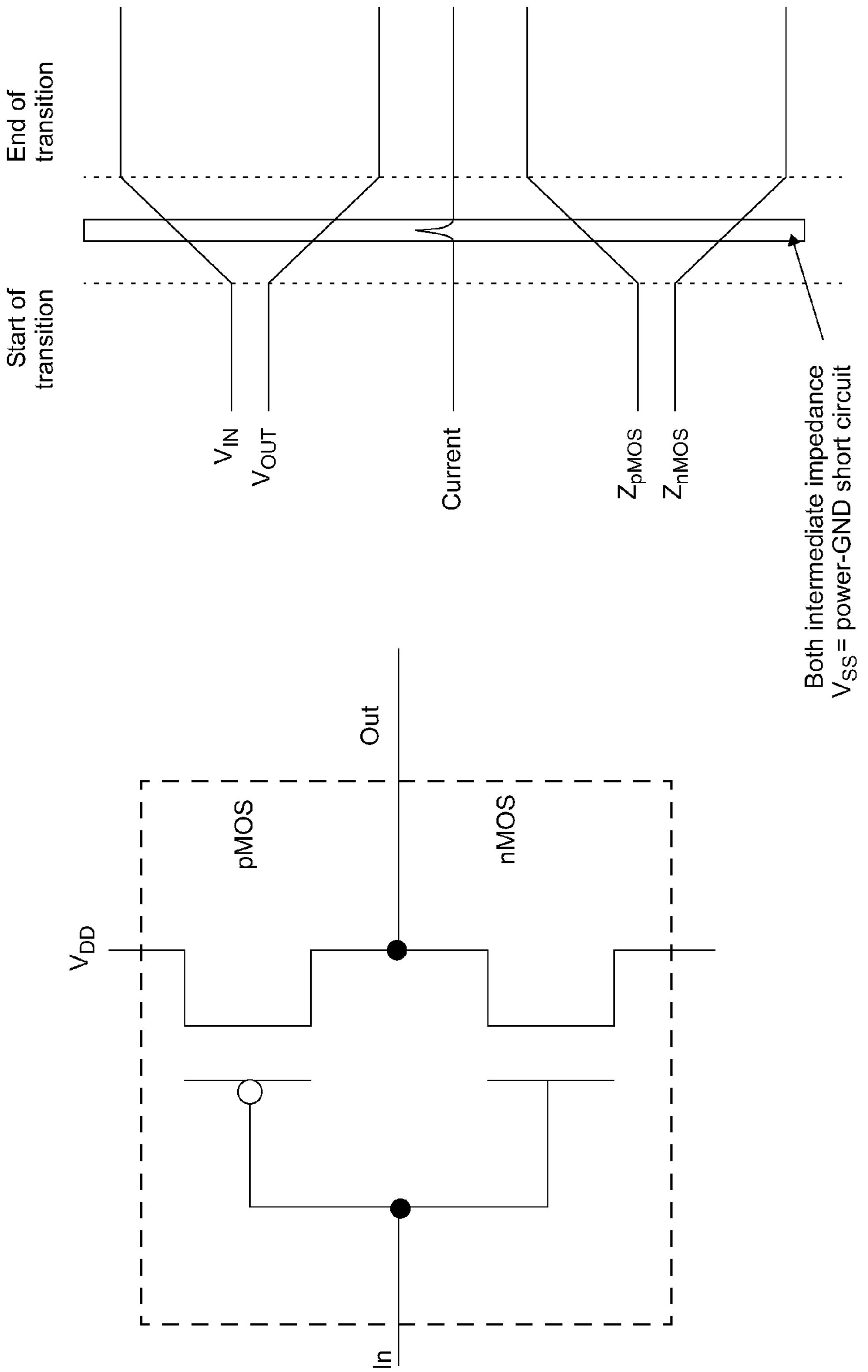

Start of transition
End of transition
V_{IN}
V_{OUT}
Current
Z_{pMOS}
Z_{nMOS}
Both intermediate impedance
V_{SS} = power-GND short circuit
V_{DD}
pMOS
nMOS
Out
In

FIG. 18

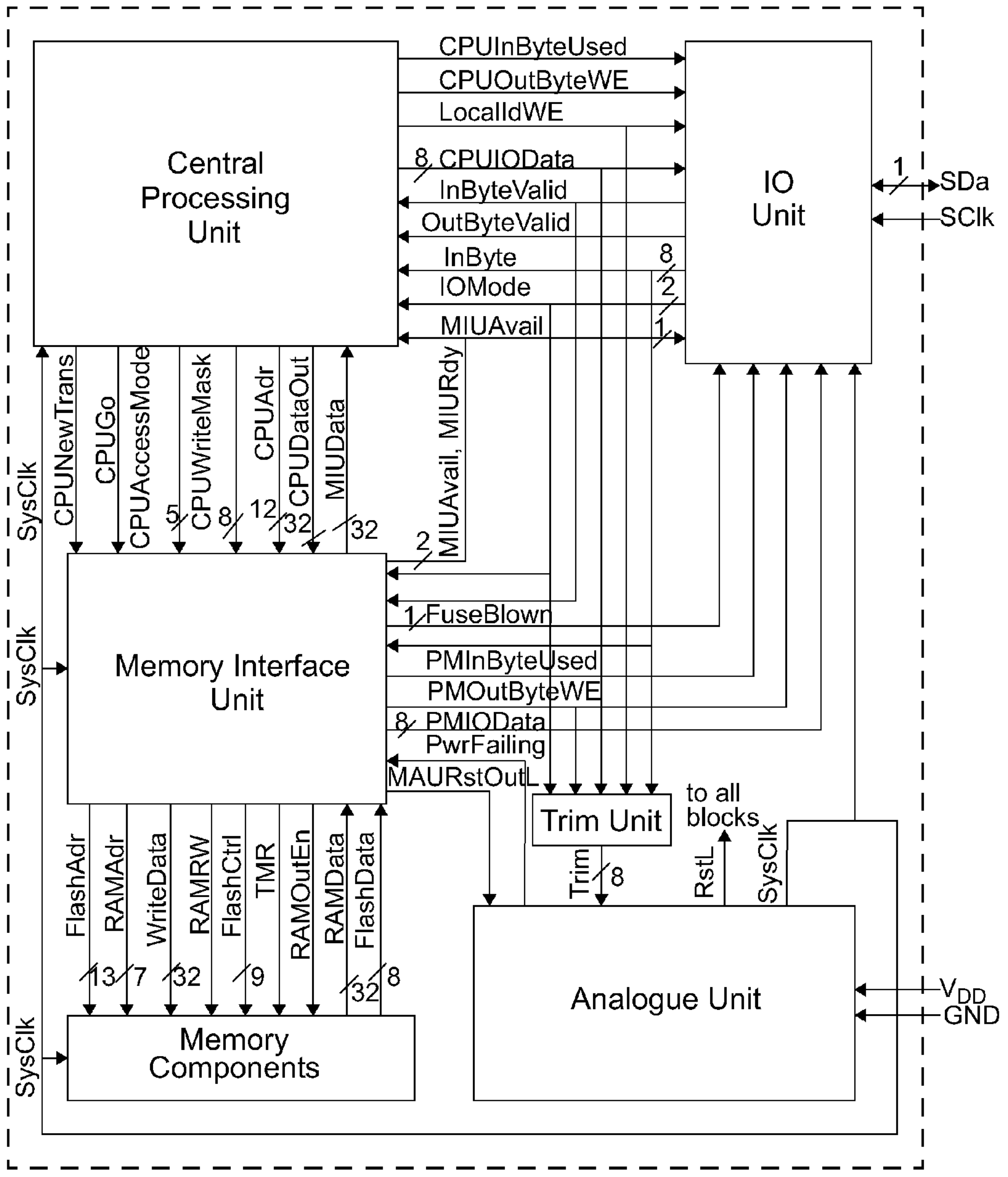

Central Processing Unit
IO Unit
Memory Interface Unit
Memory Components
Trim Unit
Analogue Unit
CPUInByteUsed
CPUOutByteWE
LocalIdWE
CPUIOData
InByteValid
OutByteValid
InByte
IOMode
MIUAvail
SDa
SClk
SysClk
CPUNewTrans
CPUGo
CPUAccessMode
CPUWriteMask
CPUAdr
CPUDataOut
MIUData
MIUAvail, MIURdy
FuseBlown
PMInByteUsed
PMOutByteWE
PMIOData
PwrFailing
MAURstOutL
FlashAdr
RAMAdr
WriteData
RAMRW
FlashCtrl
TMR
RAMOutEn
RAMData
FlashData
Trim
RstL
to all blocks
V_{DD}
GND

FIG. 19

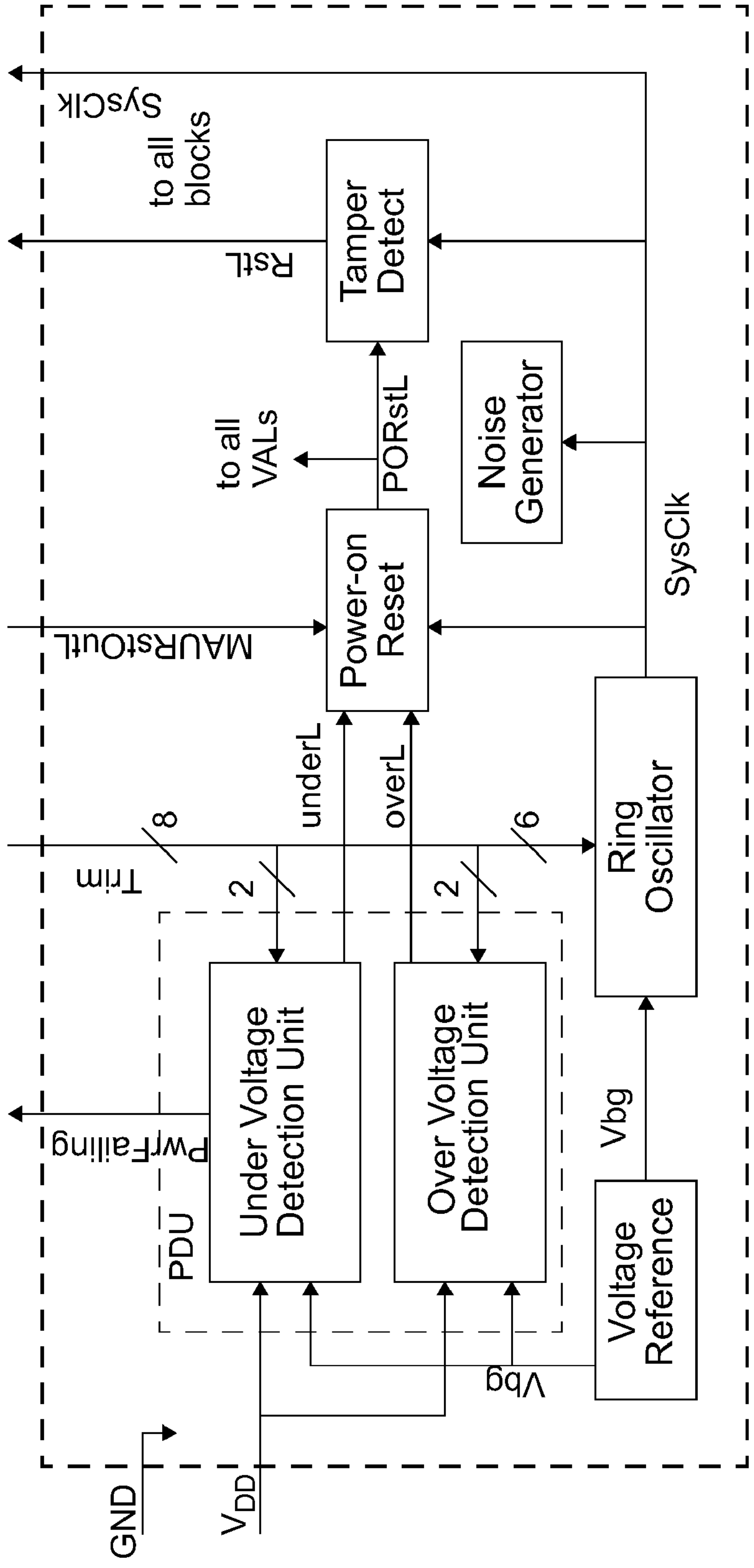

SysClk
to all blocks
RstL
Tamper Detect
to all VALs
PORstL
Noise Generator
MAURstOutL
Power-on Reset
SysClk
Trim
8
underL
overL
6
2
2
Ring Oscillator
PwrFailing
PDU
Under Voltage Detection Unit
Over Voltage Detection Unit
Vbg
Voltage Reference
Vbg
GND
V_{DD}

FIG. 20

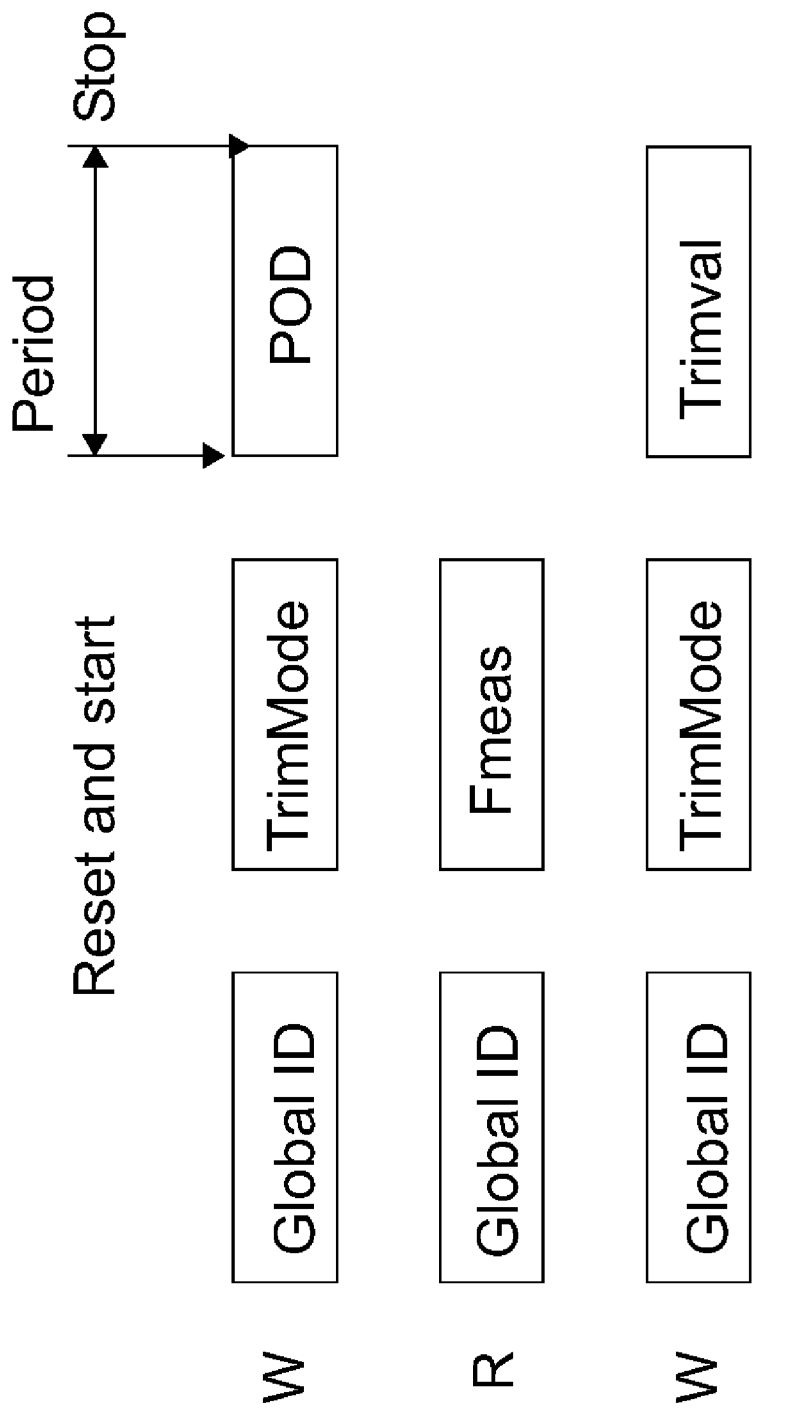

Reset and start
Period
Stop
W
Global ID
TrimMode
POD
R
Global ID
Fmeas
W
Global ID
TrimMode
Trimval

FIG. 21

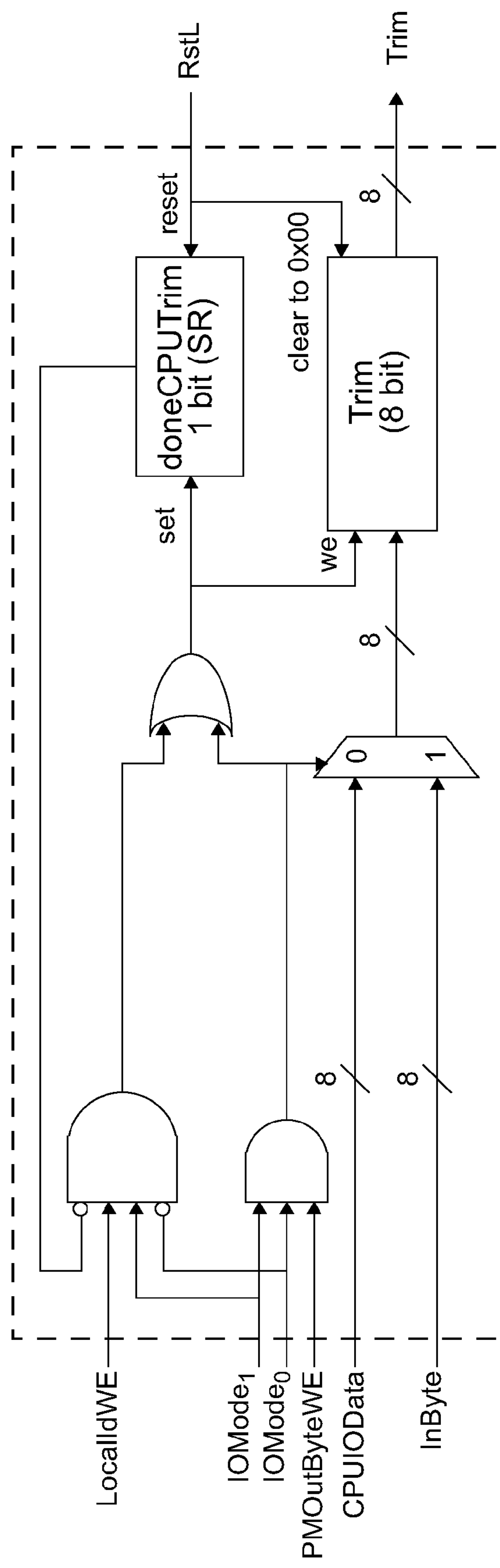

RstL
Trim
reset
clear to 0x00
8
doneCPUTrim
1 bit (SR)
Trim
(8 bit)
set
we
8
0
1
8
8
LocalIdWE
$IOMode_1$
$IOMode_0$
PMOutByteWE
CPUIOData
InByte

FIG. 22

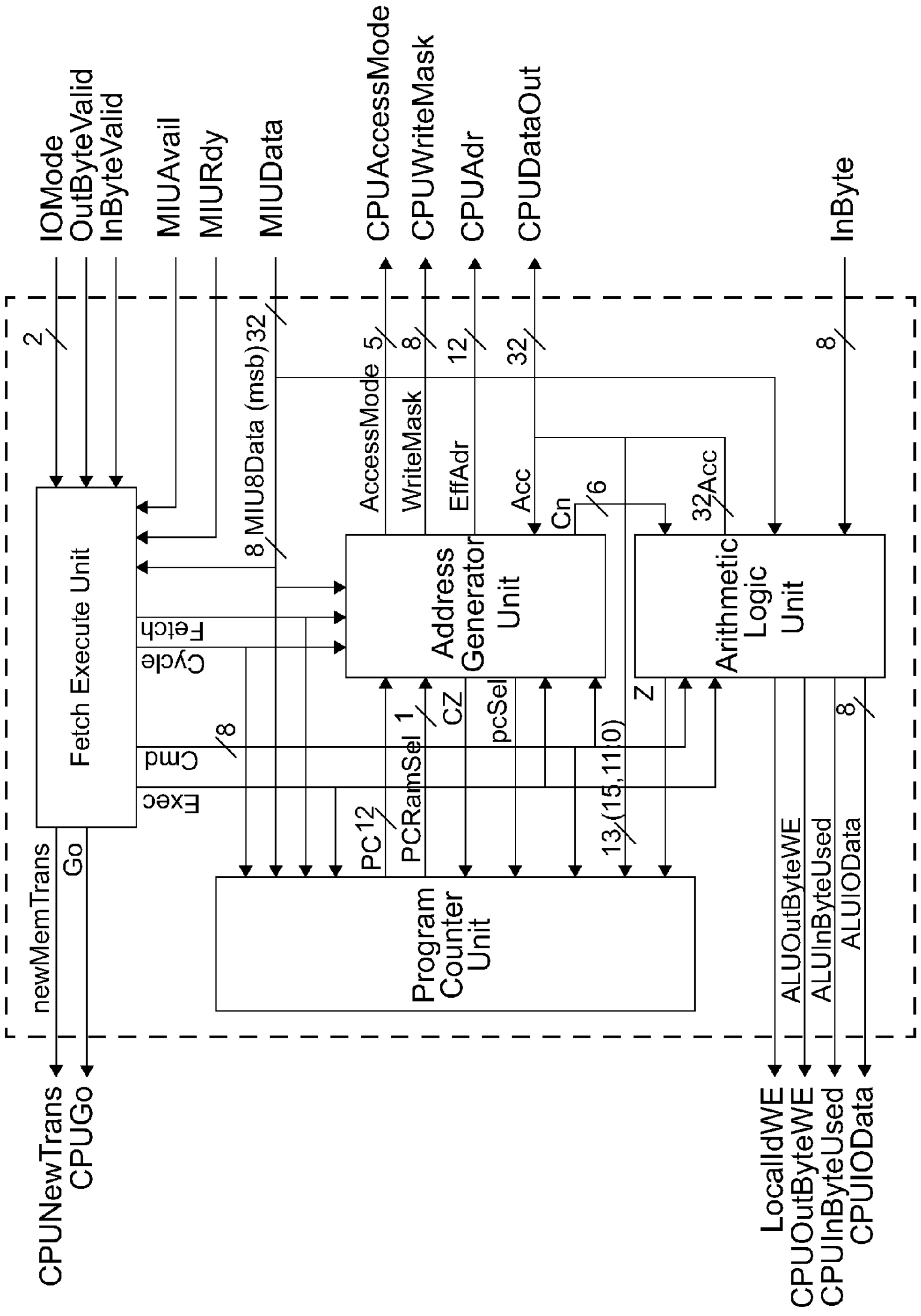

IOMode
OutByteValid
InByteValid
MIUAvail
MIURdy
MIUData
CPUAccessMode
CPUWriteMask
CPUAdr
CPUDataOut
InByte
2
32
8 MIU8Data (msb)
AccessMode 5
WriteMask 8
EffAdr 12
Acc 32
8
Fetch Execute Unit
Fetch
Cycle
Cmd
8
Exec
Address Generator Unit
Cn
6
32 Acc
Arithmetic Logic Unit
PC 12
PCRamSel 1
CZ
pcSel
13 (15,11:0)
Z
8
Program Counter Unit
newMemTrans
Go
ALUOutByteWE
ALUInByteUsed
ALUIOData
CPUNewTrans
CPUGo
LocalldWE
CPUOutByteWE
CPUInByteUsed
CPUIOData

FIG. 23

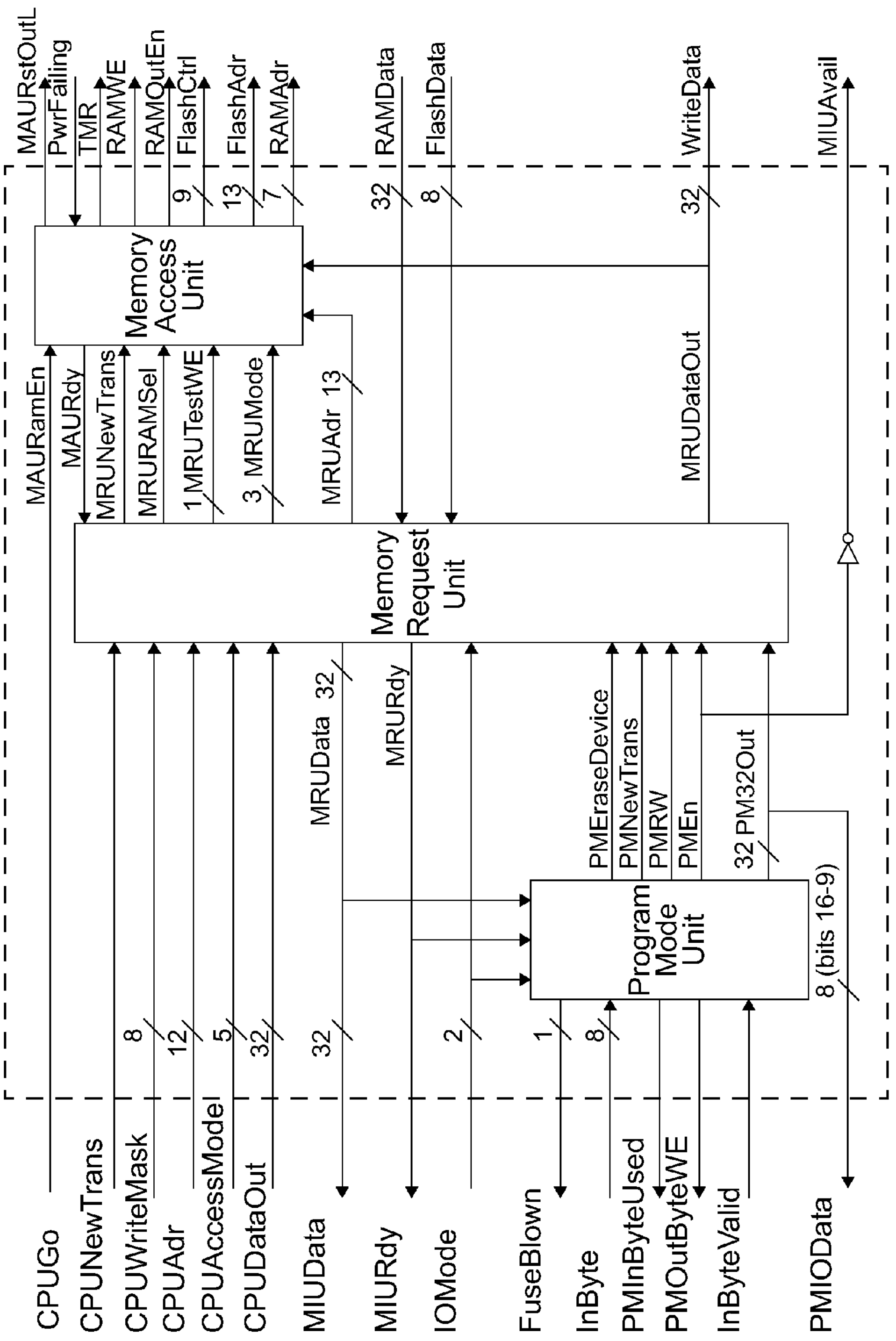

CPUGo
CPUNewTrans
CPUWriteMask
CPUAdr
CPUAccessModel
CPUDataOut
MIUData
MIURdy
IOMode
FuseBlown
InByte
PMInByteUsed
PMOutByteWE
InByteValid
PMIOData
8
12
5
32
32
2
1
8
Program Mode Unit
8 (bits 16-9)
MRUData 32
MRURdy
PMEraseDevice
PMNewTrans
PMRW
PMEn
32 PM32Out
Memory Request Unit
MAURamEn
MAURdy
MRUNewTrans
MRURAMSel
1 MRUTestWE
3 MRUMode
MRUAdr 13
MRUDataOut
Memory Access Unit
MAURstOutL
PwrFailing
TMR
RAMWE
RAMOutEn
FlashCtrl
FlashAdr
RAMAdr
RAMData
FlashData
WriteData
MIUAvail
9
13
7
32
8
32

FIG. 24

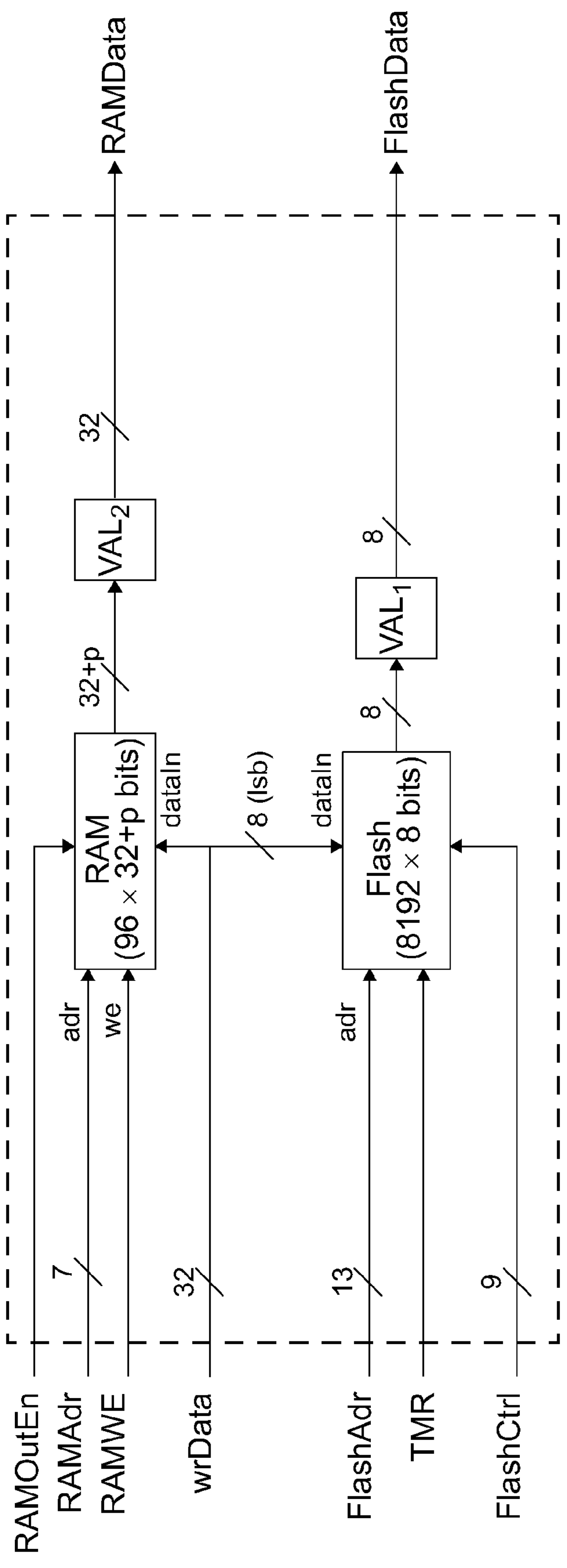

RAMOutEn
RAMAdr
RAMWE
wrData
FlashAdr
TMR
FlashCtrl
7
32
13
9
adr
we
adr
RAM (96 × 32+p bits)
dataIn
8 (lsb)
dataIn
Flash (8192 × 8 bits)
32+p
VAL2
32
8
VAL1
8
RAMData
FlashData

FIG. 25

INTEGRATED CIRCUIT FOR VALIDATING AND DECRYPTING SOFTWARE DATA

CROSS REFERENCE TO RELATED APPLICATIONS

[0001] The present application is a Continuation of U.S. application Ser. No. 10/754,536 filed Jan. 12, 2004, which is a divisional of U.S. application Ser. No. 10/727,251 filed on Dec. 2, 2003, now issued as U.S. Pat. No. 7,188,282, all of which are herein incorporated by reference.

FIELD OF INVENTION

[0002] The present invention relates to securing an integrated circuit against certain forms of security attacks.

[0003] The invention has primarily been developed for use in authentication chips used in a printer system to authenticate communications between, for example, a printer controller and other peripheral devices such as ink cartridges. However, it will be appreciated that the invention can be applied to integrated circuits in other fields in which analogous problems are faced.

BACKGROUND OF INVENTION

[0004] Manufacturing a printhead that has relatively high resolution and print-speed raises a number of problems.

[0005] Difficulties in manufacturing pagewidth printheads of any substantial size arise due to the relatively small dimensions of standard silicon wafers that are used in printhead (or printhead module) manufacture. For example, if it is desired to make an 8 inch wide pagewidth printhead, only one such printhead can be laid out on a standard 8-inch wafer, since such wafers are circular in plan. Manufacturing a pagewidth printhead from two or more smaller modules can reduce this limitation to some extent, but raises other problems related to providing a joint between adjacent printhead modules that is precise enough to avoid visible artefacts (which would typically take the form of noticeable lines) when the printhead is used. The problem is exacerbated in relatively high-resolution applications because of the tight tolerances dictated by the small spacing between nozzles.

[0006] The quality of a joint region between adjacent printhead modules relies on factors including a precision with which the abutting ends of each module can be manufactured, the accuracy with which they can be aligned when assembled into a single printhead, and other more practical factors such as management of ink channels behind the nozzles. It will be appreciated that the difficulties include relative vertical displacement of the printhead modules with respect to each other.

[0007] Whilst some of these issues may be dealt with by careful design and manufacture, the level of precision required renders it relatively expensive to manufacture printheads within the required tolerances.

[0008] It would be desirable to provide a solution to one or more of the problems associated with precision manufacture and assembly of multiple printhead modules to form a printhead, and especially a pagewidth printhead.

[0009] In some cases, it is desirable to produce a number of different printhead module types or lengths on a substrate to maximise usage of the substrate's surface area. However, different sizes and types of modules will have different numbers and layouts of print nozzles, potentially including different horizontal and vertical offsets. Where two or more modules are to be joined to form a single printhead, there is also the problem of dealing with different seam shapes between abutting ends of joined modules, which again may incorporate vertical or horizontal offsets between the modules. Printhead controllers are usually dedicated application specific integrated circuits (ASICs) designed for specific use with a single type of printhead module, that is used by itself rather than with other modules. It would be desirable to provide a way in which different lengths and types of printhead modules could be accounted for using a single printer controller.

[0010] Printer controllers face other difficulties when two or more printhead modules are involved, especially if it is desired to send dot data to each of the printheads directly (rather than via a single printhead connected to the controller). One concern is that data delivered to different length controllers at the same rate will cause the shorter of the modules to be ready for printing before any longer modules. Where there is little difference involved, the issue may not be of importance, but for large length differences, the result is that the bandwidth of a shared memory from which the dot data is supplied to the modules is effectively left idle once one of the modules is full and the remaining module or modules is still being filled. It would be desirable to provide a way of improving memory bandwidth usage in a system comprising a plurality of printhead modules of uneven length.

[0011] In any printing system that includes multiple nozzles on a printhead or printhead module, there is the possibility of one or more of the nozzles failing in the field, or being inoperative due to manufacturing defect. Given the relatively large size of a typical printhead module, it would be desirable to provide some form of compensation for one or more "dead" nozzles. Where the printhead also outputs fixative on a per-nozzle basis, it is also desirable that the fixative is provided in such a way that dead nozzles are compensated for.

[0012] A printer controller can take the form of an integrated circuit, comprising a processor and one or more peripheral hardware units for implementing specific data manipulation functions. A number of these units and the processor may need access to a common resource such as memory. One way of arbitrating between multiple access requests for a common resource is timeslot arbitration, in which access to the resource is guaranteed to a particular requestor during a predetermined timeslot.

[0013] One difficulty with this arrangement lies in the fact that not all access requests make the same demands on the resource in terms of timing and latency. For example, a memory read requires that data be fetched from memory, which may take a number of cycles, whereas a memory write can commence immediately. Timeslot arbitration does not take into account these differences, which may result in accesses being performed in a less efficient manner than might otherwise be the case. It would be desirable to provide a timeslot arbitration scheme that improved this efficiency as compared with prior art timeslot arbitration schemes.

[0014] Also of concern when allocating resources in a timeslot arbitration scheme is the fact that the priority of an access request may not be the same for all units. For example, it would be desirable to provide a timeslot arbitration scheme in which one requestor (typically the memory) is granted special priority such that its requests are dealt with earlier than would be the case in the absence of such priority.

[0015] In systems that use a memory and cache, a cache miss (in which an attempt to load data or an instruction from a cache fails) results in a memory access followed by a cache update. It is often desirable when updating the cache in this way to update data other than that which was actually missed. A typical example would be a cache miss for a byte resulting in an entire word or line of the cache associated with that byte being updated. However, this can have the effect of tying up bandwidth between the memory (or a memory manager) and the processor where the bandwidth is such that several cycles are required to transfer the entire word or line to the cache. It would be desirable to provide a mechanism for updating a cache that improved cache update speed and/or efficiency.

[0016] Most integrated circuits an externally provided signal as (or to generate) a clock, often provided from a dedicated clock generation circuit. This is often due to the difficulties of providing an onboard clock that can operate at a speed that is predictable. Manufacturing tolerances of such on-board clock generation circuitry can result in clock rates that vary by a factor of two, and operating temperatures can increase this margin by an additional factor of two. In some cases, the particular rate at which the clock operates is not of particular concern. However, where the integrated circuit will be writing to an internal circuit that is sensitive to the time over which a signal is provided, it may be undesirable to have the signal be applied for too long or short a time. For example, flash memory is sensitive to being written too for too long a period. It would be desirable to provide a mechanism for adjusting a rate of an on-chip system clock to take into account the impact of manufacturing variations on clock-speed.

[0017] One form of attacking a secure chip is to induce (usually by increasing) a clock speed that takes the logic outside its rated operating frequency. One way of doing this is to reduce the temperature of the integrated circuit, which can cause the clock to race. Above a certain frequency, some logic will start malfunctioning. In some cases, the malfunction can be such that information on the chip that would otherwise be secure may become available to an external connection. It would be desirable to protect an integrated circuit from such attacks.

[0018] In an integrated circuit comprising non-volatile memory, a power failure can result in unintentional behaviour. For example, if an address or data becomes unreliable due to falling voltage supplied to the circuit but there is still sufficient power to cause a write, incorrect data can be written. Even worse, the data (incorrect or not) could be written to the wrong memory. The problem is exacerbated with multi-word writes. It would be desirable to provide a mechanism for reducing or preventing spurious writes when power to an integrated circuit is failing.

[0019] In an integrated circuit, it is often desirable to reduce unauthorised access to the contents of memory. This is particularly the case where the memory includes a key or some other form of security information that allows the integrated circuit to communicate with another entity (such as another integrated circuit, for example) in a secure manner. It would be particularly advantageous to prevent attacks involving direct probing of memory addresses by physically investigating the chip (as distinct from electronic or logical attacks via manipulation of signals and power supplied to the integrated circuit).

[0020] It is also desirable to provide an environment where the manufacturer of the integrated circuit (or some other authorised entity) can verify or authorize code to be run on an integrated circuit.

[0021] Another desideratum would be the ability of two or more entities, such as integrated circuits, to communicate with each other in a secure manner. It would also be desirable to provide a mechanism for secure communication between a first entity and a second entity, where the two entities, whilst capable of some form of secure communication, are not able to establish such communication between themselves.

[0022] In a system that uses resources (such as a printer, which uses inks) it may be desirable to monitor and update a record related to resource usage. Authenticating ink quality can be a major issue, since the attributes of inks used by a given printhead can be quite specific. Use of incorrect ink can result in anything from misfiring or poor performance to damage or destruction of the printhead. It would therefore be desirable to provide a system that enables authentication of the correct ink being used, as well as providing various support systems secure enabling refilling of ink cartridges.

[0023] In a system that prevents unauthorized programs from being loaded onto or run on an integrated circuit, it can be laborious to allow developers of software to access the circuits during software development. Enabling access to integrated circuits of a particular type requires authenticating software with a relatively high-level key. Distributing the key for use by developers is inherently unsafe, since a single leak of the key outside the organization could endanger security of all chips that use a related key to authorize programs. Having a small number of people with high-security clearance available to authenticate programs for testing can be inconvenient, particularly in the case where frequent incremental changes in programs during development require testing. It would be desirable to provide a mechanism for allowing access to one or more integrated circuits without risking the security of other integrated circuits in a series of such integrated circuits.

[0024] In symmetric key security, a message, denoted by M, is plaintext. The process of transforming M into ciphertext C, where the substance of M is hidden, is called encryption. The process of transforming C back into M is called decryption. Referring to the encryption function as E, and the decryption function as D, we have the following identities:

$$E[M]=C$$

$$D[C]=M$$

Therefore the following identity is true:

$$D[E[M]]=M$$

A symmetric encryption algorithm is one where:

[0025] the encryption function E relies on key K_1,

[0026] the decryption function D relies on key K_2,

[0027] K_2 can be derived from K_1, and

[0028] K_1 can be derived from K_2.

[0029] In most symmetric algorithms, K_1 equals K_2. However, even if K_1 does not equal K_2, given that one key can be derived from the other, a single key K can suffice for the mathematical definition. Thus:

$$E_K[M]=C$$

$$D_K[C]=M$$

[0030] The security of these algorithms rests very much in the key K. Knowledge of K allows anyone to encrypt or decrypt. Consequently K must remain a secret for the duration of the value of M. For example, M may be a wartime message "My current position is grid position **123-456**". Once the war is over the value of M is greatly reduced, and if K is made public, the knowledge of the combat unit's position may be of no relevance whatsoever. The security of the particular symmetric algorithm is a function of two things: the strength of the algorithm and the length of the key. An asymmetric encryption algorithm is one where:

[0031] the encryption function E relies on key K_1,

[0032] the decryption function D relies on key K_2,

[0033] K_2 cannot be derived from K_1 in a reasonable amount of time, and

[0034] K_1 cannot be derived from K_2 in a reasonable amount of time.

Thus:

[0035]

$$E_{K1}[M]=C$$

$$D_{K2}[C]=M$$

[0036] These algorithms are also called public-key because one key K_1 can be made public. Thus anyone can encrypt a message (using K_1) but only the person with the corresponding decryption key (K_2) can decrypt and thus read the message. In most cases, the following identity also holds:

$$E_{K2}[M]=C$$

$$D_{K1}[C]=M$$

[0037] This identity is very important because it implies that anyone with the public key K_1 can see M and know that it came from the owner of K_2. No-one else could have generated C because to do so would imply knowledge of K_2. This gives rise to a different application, unrelated to encryption—digital signatures.

[0038] A number of public key cryptographic algorithms exist. Most are impractical to implement, and many generate a very large C for a given M or require enormous keys. Still others, while secure, are far too slow to be practical for several years. Because of this, many public key systems are hybrid—a public key mechanism is used to transmit a symmetric session key, and then the session key is used for the actual messages.

[0039] All of the algorithms have a problem in terms of key selection. A random number is simply not secure enough. The two large primes p and q must be chosen carefully—there are certain weak combinations that can be factored more easily (some of the weak keys can be tested for). But nonetheless, key selection is not a simple matter of randomly selecting 1024 bits for example. Consequently the key selection process must also be secure.

[0040] Symmetric and asymmetric schemes both suffer from a difficulty in allowing establishment of multiple relationships between one entity and a two or more others, without the need to provide multiple sets of keys. For example, if a main entity wants to establish secure communications with two or more additional entities, it will need to maintain a different key for each of the additional entities. For practical reasons, it is desirable to avoid generating and storing large numbers of keys. To reduce key numbers, two or more of the entities may use the same key to communicate with the main entity. However, this means that the main entity cannot be sure which of the entities it is communicating with. Similarly, messages from the main entity to one of the entities can be decrypted by any of the other entities with the same key. It would be desirable if a mechanism could be provided to allow secure communication between a main entity and one or more other entities that overcomes at least some of the shortcomings of prior art.

[0041] In a system where a first entity is capable of secure communication of some form, it may be desirable to establish a relationship with another entity without providing the other entity with any information related the first entity's security features. Typically, the security features might include a key or a cryptographic function. It would be desirable to provide a mechanism for enabling secure communications between a first and second entity when they do not share the requisite secret function, key or other relationship to enable them to establish trust.

[0042] A number of other aspects, features, preferences and embodiments are disclosed in the Detailed Description of the Preferred Embodiment below.

SUMMARY OF THE INVENTION

[0043] In accordance with a first aspect of the invention, there is provided a printer controller comprising an integrated circuit incorporating a processor and memory, the memory storing a set of data representing program code and/or an operating value for printer control, wherein each bit of the data is stored as a bit/inverse-bit pair in corresponding pairs of physically adjacent bit cells in the memory.

[0044] Preferably, the printer controller further includes a memory management unit configured to receive a request for the set of data and to test, during processing of the request, whether the respective pairs of physically adjacent bit-cells that correspond to the set of data contain bit/inverse-bit pairs, thereby to confirm the validity of the set of data as stored in the memory. More preferably, the memory management unit is configured to store sets of data as sets of bit/inverse-bit pairs in the memory.

[0045] Preferably, the printer controller is selectively operable in either of first and second modes, wherein in the first mode, the memory management unit is configured to receive and process a request for the set of data, and to test, during processing of the request, whether the respective pairs of physically adjacent bit-cells corresponding to the set of data contain bit/inverse-bit pairs, thereby to confirm the validity of the set of data as stored in the memory, and in the second mode, the memory management unit is configured to receive and process a request for data stored in the memory, without testing whether pairs of physically adjacent bit-cells contain bit/inverse-bit pairs.

[0046] More preferably in the first mode, the memory management unit is configured to store a set of data associated with a memory write request as a corresponding set of bit/inverse-bit pairs, each of the bit/inverse-bit pairs being physically adjacent each other, and in the second mode, the memory management unit is configured to store a set of data associated with a memory write request as the set of data without corresponding inverse-bits.

[0047] Preferably, the printer controller is configured to boot into the first mode by default. Preferably, the printer controller is configured to implement a defensive action in the event the test fails. More preferably, the defensive action includes resetting the integrated circuit.

[0048] In an alternative embodiment, the defensive reaction includes returning second data other than that the subject of the test. Preferably, the second data is a string of identical digits. Preferably, the defensive reaction is different depending upon whether the set of data represents program code or an operating value. More preferably, in the event the test fails and the set of data is an operating value, the integrated circuit is configured to replace the failed value with a substitute value. More preferably, the substitute value is selected to disrupt a program running on the integrated circuit.

[0049] Preferably, the substitute causes at least some circuitry on the integrated circuit to reset. In a preferred embodiment, in the event the test fails, the integrated circuit is permanently prevented from running software. Preferably, in the event the test fails, the integrated circuit is configured to delete from the memory some or all of the bit values associated with the set of data. More preferably, in the event the test fails, the integrated circuit is configured to delete some or all of the contents of the memory.

BRIEF DESCRIPTION OF THE DRAWINGS

[0050] Preferred and other embodiments of the invention will now be described, by way of example only, with reference to the accompanying drawings, in which:

[0051] FIG. **1** is an example of a single printer controller (hereinafter "SoPEC") A4 simplex printer system

[0052] FIG. **2** shows a SoPEC system top level partition

[0053] FIG. **3** shows a SoPEC CPU memory map (not to scale)

[0054] FIG. **4** is a block diagram of CPU

[0055] FIG. **5** shows CPU bus transactions

[0056] FIG. **6** shows a state machine for a CPU subsystem slave

[0057] FIG. **7** shows a SoPEC CPU memory map (not to scale)

[0058] FIG. **8** shows an external signal view of a memory management unit (hereinafter "MMU") sub-block partition

[0059] FIG. **9** shows an internal signal view of an MMU sub-block partition

[0060] FIG. **10** shows a DRAM write buffer

[0061] FIG. **11** shows relationship between datasets

[0062] FIG. **12** shows a validation hierarchy

[0063] FIG. **13** shows development of operating system code

[0064] FIG. **14** shows tamper detection line

[0065] FIG. **15** shows an oversize nMOS transistor layout of Tamper Detection Line

[0066] FIG. **16** shows a Tamper Detection Line

[0067] FIG. **17** shows how Tamper Detection Lines cover the Noise Generator

[0068] FIG. **18** shows a prior art FET Implementation of CMOS inverter

[0069] FIG. **19** shows a high level block diagram of QA IC

[0070] FIG. **20** shows an analogue unit

[0071] FIG. **21** shows a serial bus protocol for trimming

[0072] FIG. **22** shows a block diagram of a trim unit

[0073] FIG. **23** shows a block diagram of a CPU of the QA IC

[0074] FIG. **24** shows block diagram of an MIU

[0075] FIG. **25** shows a block diagram of memory components

DETAILED DESCRIPTION OF EMBODIMENTS

[0076] The preferred of the present invention is implemented in a printer using microelectromechanical systems (MEMS) printheads. The printer can receive data from, for example, a personal computer such as an IBM compatible PC or Apple computer. In other embodiments, the printer can receive data directly from, for example, a digital still or video camera. The particular choice of communication link is not important, and can be based, for example, on USB, Firewire, Bluetooth or any other wireless or hardwired communications protocol.

[0077] The printer incorporates a printer controller (SoPEC or Small office home office Print Engine Controller) having an ASIC (Application Specific Integrated Circuit). The SoPEC ASIC is intended to be a low cost solution for bi-lithic printhead control, replacing the multichip solutions in larger more professional systems with a single chip. The increased cost competitiveness is achieved by integrating several systems such as a modified PEC **1** printing pipeline, CPU control system, peripherals and memory sub-system onto one SoC ASIC, reducing component count and simplifying board design.

[0078] The following terms are used throughout this specification:

[0079] Bi-lithic printhead refers to printhead constructed from 2 printhead ICs;

[0080] CPU refers to CPU core, caching system and memory management unit (MMU);

[0081] ISI-Bridge chip a device with a high speed interface (such as USB2.0, Ethernet or IEEE1394) and one or more ISI interfaces. The ISI-Bridge would be the ISIMaster for each of the ISI buses it interfaces to;

[0082] ISIMaster the ISIMaster is the only device allowed to initiate communication on the Inter Sopec Interface (ISI) bus. The ISIMaster interfaces with the host;

[0083] ISISlave multi-SoPEC systems will contain one or more ISISlave SoPECs connected to the ISI bus. ISISlaves can only respond to communication initiated by the ISIMaster;

[0084] LEON refers to the LEON CPU core;

[0085] LineSyncMaster the LineSyncMaster device generates the line synchronisation pulse that all SoPECs in the system must synchronise their line outputs to;

[0086] Multi-SoPEC refers to SoPEC based print system with multiple SoPEC devices;

[0087] Netpage refers to page printed with tags (normally in infrared ink);

[0088] PEC1 refers to Print Engine Controller version 1, precursor to SoPEC used to control printheads constructed from multiple angled printhead segments;

[0089] Printhead IC single MEMS IC used to construct bi-lithic printhead;

[0090] PrintMaster the PrintMaster device is responsible for coordinating all aspects of the print operation. There may only be one PrintMaster in a system;

[0091] QA IC/Device Quality Assurance Integrated Circuit/Device;

[0092] Storage SoPEC an ISISlave SoPEC used as a DRAM store and which does not print; and

[0093] Tag refers to pattern which encodes information about its position and orientation which allow it to be optically located and its data contents read.

[0094] The SoPEC device can be used in several printer configurations and architectures. In the general sense every SoPEC based printer architecture will contain:

[0095] One or more SoPEC devices.

[0096] One or more bi-lithic printheads.

[0097] Two or more LSS busses.

[0098] Two or more QA ICs.

[0099] USB 1.1 connection to host or ISI connection to Bridge Chip.

[0100] ISI bus connection between SoPECs (when multiple SoPECs are used).

[0101] The SoPEC device contains several system on a chip (SoC) components, as well as the print engine pipeline (PEP) control application specific logic.

[0102] The PEP reads compressed page store data from the embedded memory, optionally decompresses the data and formats it for sending to the printhead. The print engine pipeline functionality includes expanding the page image, dithering the contone layer, compositing the black layer over the contone layer, rendering of Netpage tags, compensation for dead nozzles in the printhead, and sending the resultant image to the bi-lithic printhead.

[0103] SoPEC contains an embedded CPU for general purpose system configuration and management. The CPU performs page and band header processing, motor control and sensor monitoring (via the GPIO) and other system control functions. The CPU can perform buffer management or report buffer status to the host. The CPU can optionally run vendor application specific code for general print control such as paper ready monitoring and LED status update.

[0104] A 2.5 Mbyte embedded memory buffer is integrated onto the SoPEC device, of which approximately 2 Mbytes are available for compressed page store data. A compressed page is divided into one or more bands, with a number of bands stored in memory. As a band of the page is consumed by the PEP for printing a new band can be downloaded. The new band may be for the current page or the next page.

[0105] Using banding it is possible to begin printing a page before the complete compressed page is downloaded, but care must be taken to ensure that data is always available for printing or a buffer underrun may occur. An Storage SoPEC acting as a memory buffer or an ISI-Bridge chip with attached DRAM could be used to provide guaranteed data delivery.

[0106] The embedded USB 1.1 device accepts compressed page data and control commands from the host PC, and facilitates the data transfer to either embedded memory or to another SoPEC device in multi-SoPEC systems.

[0107] The printhead is constructed by abutting 2 printhead ICs together. The printhead ICs can vary in size from 2 inches to 8 inches, so to produce an A4 printhead several combinations are possible. For example two printhead ICs of 7 inches and 3 inches could be used to create a A4 printhead (the notation is 7:3). Similarly 6 and 4 combination (6:4), or 5:5 combination. For an A3 printhead it can be constructed from 8:6 or an 7:7 printhead IC combination. For photographic printing smaller printheads can be constructed.

[0108] Each SoPEC device has 2 Low Speed Serial (LSS) interfacde system buses for communication with QA devices for system authentication and ink usage accounting. The number of QA devices per bus and their position in the system is unrestricted with the exception that PRINTER_QA and INK_QA devices should be on separate LSS busses.

[0109] Each SoPEC system can have several QA devices. Normally each printing SoPEC will have an associated PRINTER_QA. Ink cartridges will contain an INK_QA IC. PRINTER_QA and INK_QA devices should be on separate LSS busses. All QA ICs in the system are physically identical with flash memory contents defining PRINTER_QA from INK_QA IC.

[0110] The Inter-SoPEC Interface (ISI) provides a communication channel between SoPECs in a multi-SoPEC system. The ISIMaster can be SoPEC device or an ISI-Bridge chip depending on the printer configuration. Both compressed data and control commands are transferred via the interface.

[0111] A device, other than a SoPEC with a USB connection, which provides print data to a number of slave SoPECs. A bridge chip will typically have a high bandwidth connection, such as USB2.0, Ethernet or IEEE1394, to a host and may have an attached external DRAM for compressed page storage. A bridge chip would have one or more ISI interfaces. The use of multiple ISI buses would allow the construction of independent print systems within the one printer. The ISI-Bridge would be the ISIMaster for each of the ISI buses it interfaces to.

[0112] The SoPEC is a page rendering engine ASIC that takes compressed page images as input, and produces decompressed page images at up to 6 channels of bi-level dot data as output. The bi-level dot data is generated for the Memjet bi-lithic printhead. The dot generation process takes account of printhead construction, dead nozzles, and allows for fixative generation.

[0113] A single SoPEC can control 2 bi-lithic printheads and up to 6 color channels at 10,000 lines/sec, equating to 30 pages per minute (at 1600 dpi). A single SoPEC can perform full-bleed printing of A3, A4 and Letter pages. The 6 channels of colored ink are the expected maximum in a consumer SOHO, or office Bi-lithic printing environment:

[0114] CMY, for regular color printing.

[0115] K, for black text, line graphics and gray-scale printing.

[0116] IR (infrared), for Netpage-enabled [5] applications.

[0117] F (fixative), to enable printing at high speed. Because the bi-lithic printer is capable of printing so fast, a fixative may be required to enable the ink to dry before the page touches the page already printed. Otherwise the pages may bleed on each other. In low speed printing environments the fixative may not be required.

[0118] SoPEC is color space agnostic. Although it can accept contone data as CMYX or RGBX, where X is an optional 4th channel, it also can accept contone data in any print color space. Additionally, SoPEC provides a mechanism for arbitrary mapping of input channels to output channels, including combining dots for ink optimization, generation of channels based on any number of other channels etc. However, inputs are typically CMYK for contone input, K for the bi-level input, and the optional Netpage tag dots are typically rendered to an infra-red layer. A fixative channel is typically generated for fast printing applications.

[0119] SoPEC is resolution agnostic. It merely provides a mapping between input resolutions and output resolutions by means of scale factors. The expected output resolution is 1600 dpi, but SoPEC actually has no knowledge of the physical resolution of the Bi-lithic printhead.

[0120] SoPEC is page-length agnostic. Successive pages are typically split into bands and downloaded into the page store as each band of information is consumed and becomes free.

[0121] SoPEC provides an interface for synchronization with other SoPECs. This allows simple multi-SoPEC solutions for simultaneous A3/A4/Letter duplex printing. However, SoPEC is also capable of printing only a portion of a page image. Combining synchronization functionality with partial page rendering allows multiple SoPECs to be readily combined for alternative printing requirements including simultaneous duplex printing and wide format printing.

[0122] The required printing rate for SoPEC is 30 sheets per minute with an inter-sheet spacing of 4 cm. To achieve a 30 sheets per minute print rate, this requires: 300 mm×63 (dot/mm)/2 sec=105.8 seconds per line, with no inter-sheet gap or 340 mm×63 (dot/mm)/2 sec=93.3 seconds per line, with a 4 cm inter-sheet gap.

[0123] A printline for an A4 page consists of 13824 nozzles across the page. At a system clock rate of 160 MHz 13824 dots of data can be generated in 86.4 seconds. Therefore data can be generated fast enough to meet the printing speed requirement. It is necessary to deliver this print data to the print-heads.

[0124] Printheads can be made up of 5:5, 6:4, 7:3 and 8:2 inch printhead combinations. Print data is transferred to both print heads in a pair simultaneously. This means the longest time to print a line is determined by the time to transfer print data to the longest print segment. There are 9744 nozzles across a 7 inch printhead.

[0125] The print data is transferred to the printhead at a rate of 106 MHz (⅔ of the system clock rate) per color plane. This means that it will take 91.9 s to transfer a single line for a 7:3 printhead configuration. So we can meet the requirement of 30 sheets per minute printing with a 4 cm gap with a 7:3 printhead combination. There are 11160 across an 8 inch printhead. To transfer the data to the printhead at 106 MHz will take 105.3 s. So an 8:2 printhead combination printing with an inter-sheet gap will print slower than 30 sheets per minute.

[0126] From the highest point of view the SoPEC device consists of 3 distinct subsystems

[0127] CPU Subsystem

[0128] DRAM Subsystem

[0129] Print Engine Pipeline (PEP) Subsystem

See FIG. **13** for a block level diagram of SoPEC.

[0130] The CPU subsystem controls and configures all aspects of the other subsystems. It provides general support for interfacing and synchronising the external printer with the internal print engine. It also controls the low speed communication to the QA ICs. The CPU subsystem contains various peripherals to aid the CPU, such as GPIO (includes motor control), interrupt controller, LSS Master and general timers. The Serial Communications Block (SCB) on the CPU subsystem provides a full speed USB 1.1 interface to the host as well as an Inter SoPEC Interface (ISI) to other SoPEC devices.

[0131] The DRAM subsystem accepts requests from the CPU, Serial Communications Block (SCB) and blocks within the PEP subsystem. The DRAM subsystem (in particular the DIU) arbitrates the various requests and determines which request should win access to the DRAM. The DIU arbitrates based on configured parameters, to allow sufficient access to DRAM for all requestors. The DIU also hides the implementation specifics of the DRAM such as page size, number of banks, refresh rates etc.

[0132] The PEP subsystem accepts compressed pages from DRAM and renders them to bi-level dots for a given print line destined for a printhead interface that communicates directly with up to 2 segments of a bi-lithic printhead.

[0133] The first stage of the page expansion pipeline is the CDU, LBD and TE. The CDU expands the JPEG-compressed contone (typically CMYK) layer, the LBD expands the compressed bi-level layer (typically K), and the TE encodes Netpage tags for later rendering (typically in IR or K ink). The output from the first stage is a set of buffers: the CFU, SFU, and TFU. The CFU and SFU buffers are implemented in DRAM.

[0134] The second stage is the HCU, which dithers the contone layer, and composites position tags and the bi-level spot0 layer over the resulting bi-level dithered layer. A number of options exist for the way in which compositing occurs. Up to 6 channels of bi-level data are produced from this stage. Note that not all 6 channels may be present on the printhead. For example, the printhead may be CMY only, with K pushed into the CMY channels and IR ignored. Alternatively, the position tags may be printed in K if IR ink is not available (or for testing purposes).

[0135] The third stage (DNC) compensates for dead nozzles in the printhead by color redundancy and error diffusing dead nozzle data into surrounding dots.

[0136] The resultant bi-level 6 channel dot-data (typically CMYK-IRF) is buffered and written out to a set of line buffers stored in DRAM via the DWU.

[0137] Finally, the dot-data is loaded back from DRAM, and passed to the printhead interface via a dot FIFO. The dot FIFO accepts data from the LLU at the system clock rate (pclk), while the PHI removes data from the FIFO and sends it to the printhead at a rate of ⅔ times the system clock rate. Looking at FIG. **13**, the various units are described here in summary form:

TABLE 1

Units within SoPEC

Subsystem	Unit Acronym	Unit Name	Description
DRAM	DIU	DRAM interface unit	Provides the interface for DRAM read and write access for the various SoPEC units, CPU and the SCB block. The DIU provides arbitration between competing units controls DRAM access.
	DRAM	Embedded DRAM	20 Mbits of embedded DRAM,
CPU	CPU	Central Processing Unit	CPU for system configuration and control
	MMU	Memory Management Unit	Limits access to certain memory address areas in CPU user mode

TABLE 1-continued

Units within SoPEC

Subsystem	Unit Acronym	Unit Name	Description
	RDU	Real-time Debug Unit	Facilitates the observation of the contents of most of the CPU addressable registers in SoPEC in addition to some pseudo-registers in realtime.
	TIM	General Timer	Contains watchdog and general system timers
	LSS	Low Speed Serial Interfaces	Low level controller for interfacing with the QA ICs
	GPIO	General Purpose IOs	General IO controller, with built-in Motor control unit, LED pulse units and de-glitch circuitry
	ROM	Boot ROM	16 KBytes of System Boot ROM code
	ICU	Interrupt Controller Unit	General Purpose interrupt controller with configurable priority, and masking.
	CPR	Clock, Power and Reset block	Central Unit for controlling and generating the system clocks and resets and powerdown mechanisms
	PSS	Power Save Storage	Storage retained while system is powered down
	USB	Universal Serial Bus Device	USB device controller for interfacing with the host USB.
	ISI	Inter-SoPEC Interface	ISI controller for data and control communication with other SoPEC's in a multi-SoPEC system
	SCB	Serial Communication Block	Contains both the USB and ISI blocks.
Print Engine Pipeline (PEP)	PCU	PEP controller	Provides external CPU with the means to read and write PEP Unit registers, and read and write DRAM in single 32-bit chunks.
	CDU	Contone decoder unit	Expands JPEG compressed contone layer and writes decompressed contone to DRAM
	CFU	Contone FIFO Unit	Provides line buffering between CDU and HCU
	LBD	Lossless Bi-level Decoder	Expands compressed bi-level layer.
	SFU	Spot FIFO Unit	Provides line buffering between LBD and HCU
	TE	Tag encoder	Encodes tag data into line of tag dots.
	TFU	Tag FIFO Unit	Provides tag data storage between TE and HCU
	HCU	Halftoner compositor unit	Dithers contone layer and composites the bi-level spot 0 and position tag dots.
	DNC	Dead Nozzle Compensator	Compensates for dead nozzles by color redundancy and error diffusing dead nozzle data into surrounding dots.
	DWU	Dotline Writer Unit	Writes out the 6 channels of dot data for a given printline to the line store DRAM
	LLU	Line Loader Unit	Reads the expanded page image from line store, formatting the data appropriately for the bi-lithic printhead.
	PHI	PrintHead Interface	Is responsible for sending dot data to the bi-lithic printheads and for providing line synchronization between multiple SoPECs. Also provides test interface to printhead such as temperature monitoring and Dead Nozzle Identification.

[0138] A number of hardware, software and protocol solutions to security issues with respect to SoPEC have been developed. These range from authorization and encryption protocols for enabling secure communication between hardware and software modules, to physical and electrical systems that protect the integrity of integrated circuits and other hardware.

[0139] It should be understood that in many cases, principles described with reference to hardware such as integrated circuits (ie, chips) can be implemented wholly or partly in software running on, for example, a computer. Mixed systems in which software and hardware (and combinations) embody various entities, modules and units can also be constructed using may of these principles, particularly in relation to authorization and authentication protocols. The particular extent to which the principles described below can be translated to or from hardware or software will be apparent to one skilled in the art, and so will not always explicitly be explained. It should also be understood that many of the techniques disclosed below have application to many fields other than printing.

[0140] A "QA IC" is a quality assurance chip can allows certain security functions and protocols to be implemented. Various authentication protocols include:

[0141] For authenticated reads, an Untrusted QA Device being a QA IC being read from, and a Trusted QA Device being a QA IC that identifies whether the data read from the Untrusted QA Device can be trusted;

[0142] For replacement of keys, a QA IC is programmed with the new key, and a Key Programmer QA Device is a factory QA IC that generates the message to program the new key; and

[0143] For upgrades of data in memory vectors, a QA IC is upgraded, and a Value or Parameter Upgrader QA Device is a QA IC that signs the upgrade value.

[0144] Any given physical QA IC will contain functionality that allows it to operate as an entity in some number of these protocols. Physical QA ICs are referred to by their location. For example, each ink cartridge may contain a QA IC referred to as an INK_QA, with all INK_QA ICs being on the same physical bus. In the same way, the QA IC inside the printer is referred to as PRINTER_QA, and will be on a separate bus to the INK_QA ICs.

[0145] When applied to a printing environment, the functional security requirements for the preferred embodiment are:

[0146] Code of QA IC owner or licensee co-existing safely with code of authorized OEMs

[0147] Chip owner/licensee operating parameters authentication

[0148] Parameters authentication for authorized OEMs

[0149] Ink usage authentication

The authentication requirements imply that:

- **[0150]** OEMs and end-users must not be able to replace or tamper with QA IC manufacturer/owner's program code or data
- **[0151]** OEMs and end-users must not be able to perform unauthorized activities for example by calling chip manufacturer/owner's code
- **[0152]** End-users must not be able to replace or tamper with OEM program code or data
- **[0153]** End-users must not be able to call unauthorized functions within OEM program code
- **[0154]** Manufacturer/owner's development program code must not be capable of running on all SoPECs.
- **[0155]** OEMs must be able to test products at their highest upgradable status, yet not be able to ship them outside the terms of their license
- **[0156]** OEMs and end-users must not be able to directly access the print engine pipeline (PEP) hardware, the LSS Master (for QA IC access) or any other peripheral block with the exception of operating system permitted GPIO pins and timers.

[0157] SoPEC includes a CPU that must run both manufacturer/owner program code and OEM program code. The execution model envisaged for SoPEC is one where Manufacturer/owner program code forms an operating system (O/S), providing services such as controlling the print engine pipeline, interfaces to communications channels etc. The OEM program code must run in a form of user mode, protected from harming the Manufacturer/owner program code. The OEM program code is permitted to obtain services by calling functions in the O/S, and the O/S may also call OEM code at specific times. For example, the OEM program code may request that the O/S call an OEM interrupt service routine when a particular GPIO pin is activated.

[0158] In addition, we may wish to permit the OEM code to directly call functions in Manufacturer/owner code with the same permissions as the OEM code. For example, the Manufacturer/owner code may provide SHA1 as a service, and the OEM could call the SHA1 function, but execute that function with OEM permissions and not manufacturer/owner permissions.

[0159] A basic requirement then, for SoPEC, is a form of protection management, whereby Manufacturer/owner and OEM program code can co-exist without the OEM program code damaging operations or services provided by the Manufacturer/owner O/S. Since services rely on SoPEC peripherals (such as USB2 Host, LSS Master, Timers etc) access to these peripherals should also be restricted to Manufacturer/owner program code only.

[0160] A particular OEM will be licensed to run a Print Engine with a particular set of operating parameters (such as print speed or quality). The OEM and/or end-user can upgrade the operating license for a fee and thereby obtain an upgraded set of operating parameters.

[0161] Neither the OEM nor end-user should be able to upgrade the operating parameters without paying the appropriate fee to upgrade the license. Similarly, neither the OEM nor end-user should be able to bypass the authentication mechanism via any program code on SoPEC. This implies that OEMs and end-users must not be able to tamper with or replace Manufacturer/owner program code or data, nor be able to call unauthorized functions within Manufacturer/owner program code.

[0162] However, the OEM must be capable of assembly-line testing the Print Engine at the upgraded status before selling the Print Engine to the end-user.

[0163] The OEM may provide operating parameters to the end-user independent of the Manufacturer/owner operating parameters. For example, the OEM may want to sell a franking machine.

[0164] The end-user should not be able to upgrade the operating parameters without paying the appropriate fee to the OEM. Similarly, the end-user should not be able to bypass the authentication mechanism via any program code on SoPEC. This implies that end-users must not be able to tamper with or replace OEM program code or data, as well as not be able to tamper with the PEP blocks or service-related peripherals.

[0165] If an end user takes the time and energy to hack the print engine and thereby succeeds in upgrading the single print engine only, yet not be able to use the same keys etc on another print engine, that is an acceptable security compromise. However it doesn't mean we have to make it totally simple or cheap for the end-user to accomplish this.

[0166] Software-only attacks are the most dangerous, since they can be transmitted via the internet and have no perceived cost. Physical modification attacks are far less problematic, since most printer users are not likely to want their print engine to be physically modified. This is even more true if the cost of the physical modification is likely to exceed the price of a legitimate upgrade.

[0167] A solution to the above requirements and others can be summarised as (which are detailed below):

[0168] Each SoPEC has a unique id

[0169] CPU with user/supervisor mode

[0170] Memory Management Unit

[0171] The unique id is not cached

[0172] SoPEC physical identification

[0173] Each SoPEC needs to contains a unique SoPEC_id of minimum size 64-bits. This SoPEC_id is used to form a symmetric key unique to each SoPEC: SoPEC_id_key. On SoPEC we make use of an additional 112-bit ECID (electronic chip ID) macro that has been programmed with a random number on a per-chip basis. Thus SoPEC_id is the 112-bit macro, and the SoPEC_id key is a 160-bit result obtained by SHA1 (SoPEC_id).

[0174] The verification of operating parameters and ink usage depends on SoPEC_id being difficult to determine. Difficult to determine means that someone should not be able to determine the id via software, or by viewing the communications between chips on the board. If the SoPEC_id is available through running a test procedure on specific test pins on the chip, then depending on the ease by which this can be done, it is likely to be acceptable.

[0175] It is important to note that in the proposed solution, compromise of the SoPEC_id leads only to compromise of the operating parameters and ink usage on this particular SoPEC. It does not compromise any other SoPEC or all inks or operating parameters in general.

[0176] It is ideal that the SoPEC_id be random, although this is unlikely to occur on standard manufacture processes for ASICs. If the id is within a small range however, it will be able to be broken by brute force. This is why 32-bits is not sufficient protection.

[0177] SoPEC contains a CPU with direct hardware support for user and supervisor modes. At present, the intended CPU is the LEON (a 32-bit processor with an instruction set according to the IEEE-1754 standard. The IEEE1754 standard is compatible with the SPARC V8 instruction set).

[0178] Manufacturer/owner (operating system) program code will run in supervisor mode, and all OEM program code will run in user mode.

[0179] SoPEC contains a Memory Management Unit (MMU) that limits access to regions of DRAM by defining read, write and execute access permissions for supervisor and user mode. Program code running in user mode is subject to user mode permission settings, and program code running in supervisor mode is subject to supervisor mode settings.

[0180] A setting of 1 for a permission bit means that type of access (e.g. read, write, execute) is permitted. A setting of 0 for a read permission bit means that that type of access is not permitted.

[0181] At reset and whenever SoPEC wakes up, the settings for all the permission bits are 1 for all supervisor mode accesses, and 0 for all user mode accesses. This means that supervisor mode program code must explicitly set user mode access to be permitted on a section of DRAM.

[0182] Access permission to all the non-valid address space should be trapped, regardless of user or supervisor mode, and regardless of the access being read, execute, or write.

[0183] Access permission to all of the valid non-DRAM address space (for example the PEP blocks) is supervisor read/write access only (no supervisor execute access, and user mode has no acccess at all) with the exception that certain GPIO and Timer registers can also be accessed by user code. These registers will require bitwise access permissions. Each peripheral block will determine how the access is restricted.

[0184] With respect to the DRAM and PEP subsystems of SoPEC, typically we would set user read/write/execute mode permissions to be 1/1/0 only in the region of memory that is used for OEM program data, I/O/1 for regions of OEM program code, and 0/0/0 elsewhere (including the trap table). By contrast we would typically set supervisor mode read/write/execute permissions for this memory to be 1/1/0 (to avoid accidentally executing user code in supervisor mode).

[0185] The SoPEC_id parameter should only be accessible in supervisor mode, and should only be stored and manipulated in a region of memory that has no user mode access.

[0186] The unique SoPEC_id needs to be available to supervisor code and not available to user code. This is taken care of by the MMU.

[0187] However the SoPEC_id must also not be accessable via the CPU's data cache or register windows. For example, if the user were to cause an interrupt to occur at a particular point in the program execution when the SoPEC_id was being manipulated, it must not be possible for the user program code to turn caching off and then access the SoPEC_id inside the data cache. This would bypass any MMU security.

[0188] The same must be true of register windows. It must not be possible for user mode program code to read or modify register settings in a supervisor program's register windows.

[0189] This means that at the least, the SoPEC_id itself must not be cacheable. Likewise, any processed form of the SoPEC_id such as the SoPEC_id key (e.g. read into registers or calculated expected results from a QA_Chip) should not be accessable by user program code.

[0190] Given that user mode program code cannot even call functions in supervisor code space, the question arises as how OEM programs can access functions, or request services. The implementation for this depends on the CPU.

[0191] On the LEON processor, the TRAP instruction allows programs to switch between user and supervisor mode in a controlled way. The TRAP switches between user and supervisor register sets, and calls a specific entry point in the supervisor code space in supervisor mode. The TRAP handler dispatches the service request, and then returns to the caller in user mode.

[0192] Use of a command dispatcher allows the O/S to provide services that filter access—e.g. a generalised print function will set PEP registers appropriately and ensure QA IC ink updates occur.

[0193] The LEON also allows supervisor mode code to call user mode code in user mode. There are a number of ways that this functionality can be implemented. It is possible to call the user code without a trap, but to return to supervisor mode requires a trap (and associated latency).

[0194] The intention is to load the Manufacturer/owner and OEM program code into SoPEC's RAM, where it can be subsequently executed. The basic SoPEC therefore, must be capable of downloading program code. However SoPEC must be able to guarantee that only authorized Manufacturer/owner boot programs can be loaded, otherwise anyone could modify the O/S to do anything, and then load that—thereby bypassing the licensed operating parameters.

[0195] Authentication of program code and data is performed using asymmetric (public-key) digital signatures and without using a QA IC.

[0196] Assuming some data has been already downloaded and a 160-bit signature into eDRAM, the boot loader needs to perform the following tasks:

- **[0197]** perform SHA-1 on the downloaded data to calculate a digest localDigest
- **[0198]** perform asymmetric decryption on the downloaded signature (160-bits) using an asymmetric public key to obtain authorizedDigest
- **[0199]** If authorizedDigest is the PKCS#1 (patent free) form of localDigest, then the down-loaded data is authorized (the signature must have been signed with the asymmetric private key) and control can then be passed to the downloaded data

[0200] Asymmetric decryption is used instead of symmetric decryption because the decrypting key must be held in SoPEC's ROM. If symmetric private keys are used, the ROM can be probed and the security is compromised. The procedure requires the following data item:

[0201] boot0key=an n-bit asymmetric public key

The procedure also requires the following two functions:

- **[0202]** SHA-1=a function that performs SHA-1 on a range of memory and returns a 160-bit digest
- **[0203]** decrypt=a function that performs asymmetric decryption of a message using the passed-in key.

[0204] The length of the key will depend on the asymmetric algorithm chosen. The key must provide the equivalent protection of the entire QA Chip system—if the Manufacturer/owner O/S program code can be bypassed, then it is equivalent to the QA Chip keys being compromised. In fact it is worse because it would compromise Manufacturer/owner operating parameters, OEM operating parameters, and ink authentication by software downloaded off the net (e.g. from some hacker).

[0205] In the case of RSA, a 2048-bit key is required to match the 160-bit symmetric-key security of the QA Chip. In the case of ECDSA, a key length of 132 bits is likely to suffice. There is no advantage to storing multiple keys in SoPEC and having the external message choose which key to validate against, because a compromise of any key allows the external user to always select that key. There is also no particular advantage to having the boot mechanism select the key (e.g. one for USB-based booting and one for external ROM booting) a compromise of the external ROM booting key is enough to compromise all the SoPEC systems.

[0206] However, there are advantages in having multiple keys present in the boot ROM and having a wire-bonding option on the pads select which of the keys is to be used. Ideally, the pads would be connected within the package, and the selection is not available via external means once the die has ben packaged. This means we can have different keys for different application areas (e.g. different uses of the chip), and if any particular SoPEC key is compromised, the die could be kept constant and only the bonding changed. Note that in the worst case of all keys being compromised, it may be economically feasible to change the boot0key value in SoPEC's ROM, since this is only a single mask change, and would be easy to verify and characterize.

Therefore the entire security of SoPEC is based on keeping the asymmetric private key paired to boot0key secure. The entire security of SoPEC is also based on keeping the program that signs (i.e. authorizes) datasets using the asymmetric private key paired to boot0key secure.

It may therefore be reasonable to have multiple signatures (and hence multiple signature programs) to reduce the chance of a single point of weakness by a rogue employee. Note that the authentication time increases linearly with the number of signatures, and requires a 2048-bit public key in ROM for each signature.

[0207] Given that test programs, evaluation programs, and Manufacturer/owner O/S code needs to be written and tested, and OEM program code etc. also needs to be tested, it is not secure to have a single authentication of a monolithic dataset combining Manufacturer/owner O/S, non-O/S, and OEM program code—we certainly don't want OEMs signing Manufacturer/owner program code, and Manufacturer/owner shouldn't have to be involved with the signing of OEM program code.

[0208] Therefore we require differing levels of authentication and therefore a number of keys, although the procedure for authentication is identical to the first—a section of program code contains the key and procedure for authenticating the next.

[0209] This method allows for any hierarchy of authentication, based on a root key of boot0key. For example, assume that we have the following entities:

- **[0210]** QACo, Manufacturer/owner's QA/key company. Knows private version of boot0key, and owner of security concerns.
- **[0211]** SoPECCo, Manufacturer/owner's SoPEC hardware/software company. Supplies SoPEC ASICs and SoPEC O/S printing software to a ComCo.
- **[0212]** ComCo, a company that assembles Print Engines from SoPECs, Memjet printheads etc, customizing the Print Engine for a given OEM according to a license
- **[0213]** OEM, a company that uses a Print Engine to create a printer product to sell to the end-users. The OEM would supply the motor control logic, user interface, and casing.

The levels of authentication hierarchy are as follows:

- **[0214]** QACo writes the boot ROM, agenerates dataset1, consisting of a boot loader program that loads and validates dataset2 and QACo's asymmetric public boot1key. QACo signs dataset0 with the asymmetric private boot0key.
- **[0215]** SoPECCo generates dataset1, consisting of the print engine security kernel O/S (which incorporates the security-based features of the print engine functionality) and the ComCo's asymmetric public key. Upon a special "formal release" request from SoPECCo, QACo signs dataset0 with QACo's asymmetric private boot0key key. The print engine program code expects to see an operating parameter block signed by the ComCo's asymmetric private key.
- **[0216]** The ComCo generates dataSet3, consisting of dataset1 plus dataset2, where dataset2 is an operating parameter block for a given OEM's print engine licence (according to the print engine license arrangement) signed with the ComCo's asymmetric private key. The operating parameter block (dataset2) would contain valid print speed ranges, a PrintEngineLicenseId, and the OEM's asymmetric public key. The ComCo can generate as many of these operating parameter blocks for any number of Print Engine Licenses, but cannot write or sign any supervisor O/S program code.
- **[0217]** The OEM would generate dataset5, consisting of dataset3 plus dataset4, where dataset4 is the OEM program code signed with the OEM's asymmetric private key. The OEM can produce as many versions of dataset5

as it likes (e.g. for testing purposes or for updates to drivers etc) and need not involve Manufacturer/owner, QACo, or ComCo in any way.

[0218] The relationship is shown in FIG. **11**.

[0219] When the end-user uses dataset5, SoPEC itself validates dataset1 via the boot0key mechanism. Once dataset1 is executing, it validates dataset2, and uses dataset2 data to validate dataset4. The validation hierarchy is shown in FIG. **12**.

[0220] If a key is compromised, it compromises all subsequent authorizations down the hierarchy. In the example from above (and as illustrated in FIG. **326**) if the OEM's asymmetric private key is compromised, then O/S program code is not compromised since it is above OEM program code in the authentication hierarchy. However if the ComCo's asymmetric private key is compromised, then the OEM program code is also compromised. A compromise of boot0key compromises everything up to SoPEC itself, and would require a mask ROM change in SoPEC to fix.

[0221] The hierarchical boot procedure gives a hierarchy of protection in a final shipped product. It is also desirable to use a hierarchy of protection during software development within Manufacturer/owner.

[0222] For a program to be downloaded and run on SoPEC during development, it will need to be signed. In addition, we don't want to have to sign each and every Manufacturer/owner development code with the boot0key, as it creates the possibility of any developmental (including buggy or rogue) application being run on any SoPEC.

[0223] Therefore QACo needs to generate/create a special intermediate boot loader, signed with boot0key, that performs the exact same tasks as the normal boot loader, except that it checks the SoPECid to see if it is a specific SoPECid (or set of SoPECids). If the SoPEC_id is in the valid set, then the developmental boot loader validates dataset2 by means of its length and a SHA-1 digest of the developmental code[1], and not by a further digital signature. The QACo can give this boot loader to the software development team within Manufacturer/owner. The software team can now write and run any program code, and load the program code using the development boot loader. There is no requirement for the subsequent software program (i.e. the developmental program code) to be signed with any key since the programs can only be run on the particular SoPECs.

[1]The SHA-1 digest is to allow the total program load time to simulate the running time of the normal boot loader running on a non-developmental version of the program.

[0224] If the developmental boot loader (and/or signature generator) were compromised, or any of the developmental programs were compromised, the worst situation is that an attacker could run programs on that particular set of SoPECs, and on no others.

[0225] This should greatly reduce the possibility of erroneous programs signed with boot0key being available to an attacker (only official releases are signed by boot0key), and therefore reduces the possibility of a Manufacturer/owner employee intentionally or inadvertently creating a back door for attackers. The relationship is shown below in FIG. **13**.

[0226] Theoretically the same kind of hierarchy could also be used to allow OEMs to be assured that their program code will only work on specific SoPECs, but this is unlikely to be necessary, and is probably undesirable.

[0227] It is possible that errors in supervisor program code (e.g. the operating system) could allow attackers to subvert the program in SoPEC and gain supervisor control. To reduce the impact of this kind of attack, it is possible to allocate some bits of the SoPEC_id to form some kind of date. The granularity of the date could be as simple as a single bit that says the date is obtained from the regular IBM ECID, or it could be 6 bits that give 10 years worth of 3-month units.

[0228] The first step of the program loaded by boot loader **0** could check the SoPEC_id date, and run or refuse to run appropriately. The Manufacturer/owner driver or OS could therefore be limited to run on SoPECs that are manufactured up until a particular date.

[0229] This means that the OEM would require a new version of the OS for SoPECs after a particular date, but the new driver could be made to work on all previous versions of SoPEC.

[0230] The function simply requires a form of date, whose granularity for working can be determined by agreement with the OEM.

[0231] For example, suppose that SoPECs are supplied with 3-month granularity in their date components. Manufacturer/owner could ship a version of the OS that works for any SoPEC of the date (i.e. on any chip), or for all SoPECs manufactured during the year etc. The driver issued the next year could work with all SoPECs up until that years etc. In this way the drivers for a chip will be backwards compatible, but will be deliberately not forwards-compatible. It allows the downloading of a new driver with no problems, but it protects against bugs in one years's driver OS from being used against future SoPECs.

[0232] Note that the phasing in of a new OS doesn't have to be at the same time as the hardware. For example, the new OS can come in 3 months before the hardware that it supports. However once the new SoPECs are being delivered, the OEM must not ship the older driver with the newer SoPECs, for the old driver will not work on the newer SoPECs. Basically once the OEM has received the new driver, they should use that driver for all SoPEC systems from that point on (old SoPECs will work with the new driver).

[0233] This date-limiting feature would most likely be using a field in the ComCo specified operating parameters, so it allows the SoPEC to use date-checking in addition to additional QA Chip related parameter checking (such as the OEM's PrintEngineLicenseId etc).

[0234] A variant on this theme is a date-window, where a start-date and end-date are specified (as relating to SoPEC manufacture, not date of use).

[0235] Operating parameters need to be considered in terms of Manufacturer/owner operating parameters and OEM operating parameters. Both sets of operating parameters are stored on the PRINTER_QA chip (physically located inside the printer). This allows the printer to maintain parameters regardless of being moved to different computers, or a loss/replacement of host O/S drivers etc.

[0236] On PRINTER_QA, memory vector M_0 contains the upgradable operating parameters, and memory vectors M_{1+} contains any constant (non-upgradable) operating parameters. Considering only Manufacturer/owner operating parameters for the moment, there are actually two problems:

[0237] a. setting and storing the Manufacturer/owner operating parameters, which should be authorized only by Manufacturer/owner

[0238] b. reading the parameters into SoPEC, which is an issue of SoPEC authenticating the data on the PRINTER_QA chip since we don't trust PRINTER_QA.

The PRINTER_QA chip therefore contains the following symmetric keys:

[0239] K_0=PrintEngineLicense_key. This key is constant for all SoPECs supplied for a given print engine license agreement between an OEM and a Manufacturer/owner ComCo. K_0 has write permissions to the Manufacturer/owner upgradeable region of M_0 on PRINTER_QA.

[0240] K_1=SoPEC_id key. This key is unique for each SoPEC (see Section 3.1), and is known only to the SoPEC and PRINTER_QA. K_1 does not have write permissions for anything.

[0241] K_0 is used to solve problem (a). It is only used to authenticate the actual upgrades of the operating parameters. Upgrades are performed using a standard upgrade protocol, with PRINTER_QA acting as the ChipU, and the external upgrader acting as the ChipS.

[0242] K_1 is used by SoPEC to solve problem (b). It is used to authenticate reads of data (i.e. the operating parameters) from PRINTER_QA. The procedure follows a standard authenticated read protocol, with PRINTER_QA acting as ChipR, and the embedded supervisor software on SoPEC acting as ChipT. The authenticated read protocol requires the use of a 160-bit nonce, which is a pseudo-random number. This creates the problem of introducing pseudo-randomness into SoPEC that is not readily determinable by OEM programs, especially given that SoPEC boots into a known state. One possibility is to use the same random number generator as in the QA Chip (a 160-bit maximal-lengthed linear feedback shift register) with the seed taken from the value in the WatchDogTimer register in SoPEC's timer unit when the first page arrives.

[0243] Note that the procedure for verifying reads of data from PRINTER_QA does not rely on Manufacturer/owner's key K_0. This means that precisely the same mechanism can be used to read and authenticate the OEM data also stored in PRINTER_QA. Of course this must be done by Manufacturer/owner supervisor code so that SoPEC_id key is not revealed.

[0244] If the OEM also requires upgradable parameters, we can add an extra key to PRINTER_QA, where that key is an OEM_key and has write permissions to the OEM part of M_0. In this way, K_1 never needs to be known by anyone except the SoPEC and PRINTER_QA.

[0245] Each printing SoPEC in a multi-SoPEC system need access to a PRINTER_QA chip that contains the appropriate SoPEC_id key to validate ink useage and operating parameters. This can be accomplished by a separate PRINTER_QA for each SoPEC, or by adding extra keys (multiple SoPEC_id keys) to a single PRINTER_QA.

[0246] However, if ink usage is not being validated (e.g. if print speed were the only Manufacturer/owner upgradable parameter) then not all SoPECs require access to a PRINTER_QA chip that contains the appropriate SoPEC_id key. Assuming that OEM program code controls the physical motor speed (different motors per OEM), then the PHI within the first (or only) front-page SoPEC can be programmed to accept (or generate) line sync pulses no faster than a particular rate. If line syncs arrived faster than the particular rate, the PHI would simply print at the slower rate. If the motor speed was hacked to be fast, the print image will appear stretched.

Manufacturer/owner operating parameters include such items as print speed, print quality etc. and are tied to a license provided to an OEM. These parameters are under Manufacturer/owner control. The licensed Manufacturer/owner operating parameters are typically stored in the PRINTER_QA.

[0247] However there are situations when it is desirable to have a floating upgrade to a license, for use on a printer of the user's choice. For example, OEMs may sell a speed-increase license upgrade that can be plugged into the printer of the user's choice. This form of upgrade can be considered a floating upgrade in that it upgrades whichever printer it is currently plugged into. This dongle is referred to as ADDITIONAL_PRINTER_QA. The software checks for the existence of an ADDITIONAL_PRINTER_QA, and if present the operating parameters are chosen from the values stored on both QA chips.

[0248] The basic problem of authenticating the additional operating parameters boils down to the problem that we don't trust ADDITIONAL_PRINTER_QA. Therefore we need a system whereby a given SoPEC can perform an authenticated read of the data in ADDITIONAL_PRINTER_QA. The SoPEC_id key is not written to a key in the ADDITIONAL_PRINTER_QA because:

[0249] then it will be tied specifically to that SoPEC, and the primary intention of the ADDITIONAL_PRINTER_QA is that it be floatable;

[0250] the ink cartridge would then not work in another printer since the other printer would not know the old SoPEC_id key (knowledge of the old key is required in order to change the old key to a new one).

[0251] updating keys is not power-safe (i.e. if at the user's site, power is removed mid-update, the ADDITIONAL_PRINTER_QA could be rendered useless)

The proposed solution is to let ADDITIONAL_PRINTER_QA have two keys:

[0252] K_0=FloatingPrintEngineLicense_key. This key has the same function as the PrintEngineLicense_key in the PRINTER_QA[2] in that K_0 has write permissions to the Manufacturer/owner upgradeable region of M_0 on ADDITIONAL_PRINTER_QA.

[2]This can be identical to PrintEngineLicense_key in the PRINTER_QA if it is desirable (unlikely) that upgraders can function on PRINTER_QAs as well as ADDITIONAL_PRINTER_QAs

[0253] K_1=UseExtParmsLicense_key. This key is constant for all of the ADDITIONAL_PRINTER_QAs for a given license agreement between an OEM and a Manufacturer/owner ComCo (this is not the same key as PrintEngineLicense_key which is stored as K_0 in PRINTER_QA). K_1 has no write permissions to anything.

[0254] K_0 is used to allow writes to the various fields containing operating parameters in the ADDITIONAL_PRINTER_QA. These writes/upgrades are performed using the standard upgrade protocol, with ADDITIONAL_PRINTER_QA acting as the ChipU, and the external upgrader acting as the ChipS. The upgrader (ChipS) also needs to check the appropriate licensing parameters such as OEM_Id for validity.

[0255] K_1 is used to allow SoPEC to authenticate reads of the ink remaining and any other ink data. This is accomplished by having the same UseExtParmsLicense_key within PRINTER_QA (e.g. in K_2), also with no write permissions. i.e:

[0256] PRINTER_QA.K_2=UseExtParmsLicense_key. This key is constant for all of the PRINTER_QAs for a given license agreement between an OEM and a Manufacturer/owner ComCo. K_2 has no write permissions to anything.

This means there are two shared keys, with PRINTER_QA sharing both, and thereby acting as a bridge between INK_QA and SoPEC.

[0257] UseExtParmsLicense_key is shared between PRINTER_QA and ADDITIONAL_PRINTER_QA

[0258] SoPEC_id key is shared between SoPEC and PRINTER_QA

[0259] All SoPEC has to do is do an authenticated read from ADDITIONAL_PRINTER_QA, pass the data/signature to PRINTER_QA, let PRINTER_QA validate the data/signature, and get PRINTER_QA to produce a similar signature based on the shared SoPEC_id key. It can do so using the Translate function. SoPEC can then compare PRINTER_QA's signature with its own calculated signature (i.e. implement a Test function in software on SoPEC), and if the signatures match, the data from ADDITIONAL_PRINTER_QA must be valid, and can therefore be trusted. Once the data from ADDITIONAL_PRINTER_QA is known to be trusted, the various operating parameters such as OEM_Id can be checked for validity.

Tying a QA_IC to be used only on a specific SoPEC can be easily accomplished by writing the PRINTER_QA's chipId (unique serial number) into an appropriate M_0 field on the ADDITIONAL_PRINTER_QA. The system software can detect the match and function appropriately. If there is no match, the software can ignore the data read from the ADDITIONAL_PRINTER_QA.

[0260] Although it is also possible to store the SoPEC_id key in one of the keys within the dongle, this must be done in an environment where power will not be removed partway through the key update process (if power is removed during the key update there is a possibility that the dongle QA Chip may be rendered unusable, although this can be checked for after the power failure).

[0261] Although an OEM should only be able sell the licensed operating parameters for a given Print Engine, they must be able to assembly-line test[3] or service/test the Print Engine with a different set of operating parameters e.g. a maximally upgraded Print Engine. Several different mechanisms can be employed to allow OEMs to test the upgraded capabilities of the Print Engine. At present it is unclear exactly what kind of assembly-line tests would be performed.

[3]This section is referring to assembly-line testing rather than development testing. An OEM can maximally upgrade a given Print Engine to allow developmental testing of their own OEM program code & mechanics.

[0262] The simplest solution is to use an ADDITIONAL_PRINTER_QA (i.e. special dongle PRINTER_QA as described in Section 3.6.5.1). The ADDITIONAL_PRINTER_QA would contain the operating parameters that maximally upgrade the printer as long as the dongle is connected to the SoPEC. The exact connection may be directly electrical (e.g. via the standard QA Chip connections) or may be over the USB connection to the printer test host depending on the nature of the test. The exact preferred connection is yet to be determined.

[0263] In the testing environment, the ADDITIONAL_PRINTER_QA also requires a numberOfImpressions field inside M_0, which is writeable by K_0. Before the SoPEC prints a page at the higher speed, it decrements the numberOfImpressions counter, performs an authenticated read to ensure the count was decremented, and then prints the page. In this way, the total number of pages that can be printed at high speed is reduced in the event of someone stealing the ADDITIONAL_PRINTER_QA device. It also means that multiple test machines can make use of the same ADDITIONAL_PRINTER_QA.

[0264] Manufacturer/owner O/S program code contains the OEM's asymmetric public key to ensure that the subsequent OEM program code is authentic—i.e. from the OEM. However given that SoPEC only contains a single root key, it is theoretically possible for different OEM's applications to be run identically physical Print Engines i.e. printer driver for OEM_1 run on an identically physical Print Engine from OEM_2.

[0265] To guard against this, the Manufacturer/owner O/S program code contains a PrintEngineLicense_id code (e.g. 16 bits) that matches the same named value stored as a fixed operating parameter in the PRINTER_QA (i.e. in M_{1+}). As with all other operating parameters, the value of PrintEngineLicense_id is stored in PRINTER_QA (and any ADDITIONAL_PRINTER_QA devices) at the same time as the other various PRINTER_QA customizations are being applied, before being shipped to the OEM site. In this way, the OEMs can be sure of differentiating themselves through software functionality.

[0266] The Manufacturer/owner O/S must perform ink authentication during prints. Ink usage authentication makes use of counters in SoPEC that keep an accurate record of the exact number of dots printed for each ink.

[0267] The ink amount remaining in a given cartridge is stored in that cartridge's INK_QA chip. Other data stored on the INK_QA chip includes ink color, viscosity, Memjet firing pulse profile information, as well as licensing parameters such as OEM_Id, inkType, InkUsageLicense_Id, etc. This information is typically constant, and is therefore likely to be stored in M_{1+} within INK_QA.

[0268] Just as the Print Engine operating parameters are validated by means of PRINTER_QA, a given Print Engine license may only be permitted to function with specifically licensed ink. Therefore the software on SoPEC could contain a valid set of ink types, colors, OEM Ids, InkUsageLicense_Ids etc. for subsequent matching against the data in the INK_QA.

[0269] SoPEC must be able to authenticate reads from the INK_QA, both in terms of ink parameters as well as ink remaining. To authenticate ink a number of steps must be taken:

[0270] restrict access to dot counts

[0271] authenticate ink usage and ink parameters via INK_QA and PRINTER_QA

[0272] broadcast ink dot usage to all SoPECs in a multi-SoPEC system

[0273] Regarding restricting access to dot counts, since the dot counts are accessed via the PHI in the PEP section of SoPEC, access to these registers (and more generally all PEP registers) must be only available from supervisor mode, and not by OEM code (running in user mode). Otherwise it might be possible for OEM program code to clear dot counts before authentication has occurred.

[0274] Regarding authenticating ink usage and ink parameters via INK_QA and PRINTER_QA, the basic problem of authentication of ink remaining and other ink data boils down to the problem that we don't trust INK_QA. Therefore how can a SoPEC know the initial value of ink (or the ink parameters), and how can a SoPEC know that after a write to the INK_QA, the count has been correctly decremented.

[0275] Taking the first issue, which is determining the initial ink count or the ink parameters, we need a system whereby a given SoPEC can perform an authenticated read of the data in INK_QA. The SoPEC_id key cannot be written to the INK_QA for two reasons:

[0276] updating keys is not power-safe (i.e. if power is removed mid-update, the INK_QA could be rendered useless)

[0277] the ink cartridge would then not work in another printer since the other printer would not know the old SoPEC_id key (knowledge of the old key is required in order to change the old key to a new one).

The proposed solution is to let INK_QA have two keys:

- **[0278]** K_0=SupplyInkLicense_key. This key is constant for all ink cartridges for a given ink supply agreement between an OEM and a Manufacturer/owner ComCo (this is not the same key as PrintEngineLicense_key which is stored as K_0 in PRINTER_QA). K_0 has write permissions to the ink remaining regions of M_0 on INK_QA.
- **[0279]** K_1=UseInkLicense_key. This key is constant for all ink cartridges for a given ink usage agreement between an OEM and a Manufacturer/owner ComCo (this is not the same key as PrintEngineLicense_key which is stored as K_0 in PRINTER_QA). K_1 has no write permissions to anything.

[0280] K_0 is used to authenticate the actual upgrades of the amount of ink remaining (e.g. to fill and refill the amount of ink). Upgrades are performed using the standard upgrade protocol, with INK_QA acting as the ChipU, and the external upgrader acting as the ChipS. The fill and refill upgrader (ChipS) also needs to check the appropriate ink licensing parameters such as OEM_Id, InkType and InkUsageLicense_Id for validity.

[0281] K_1 is used to allow SoPEC to authenticate reads of the ink remaining and any other ink data. This is accomplished by having the same UseInkLicense_key within PRINTER_QA (e.g. in K_2 or K_3), also with no write permissions.

[0282] This means there are two shared keys, with PRINTER_QA sharing both, and thereby acting as a bridge between INK_QA and SoPEC.

[0283] UseInkLicense_key is shared between INK_QA and PRINTER_QA

[0284] SoPEC_id key is shared between SoPEC and PRINTER_QA

[0285] All SoPEC has to do is do an authenticated read from INK_QA, pass the data/signature to PRINTER_QA, let PRINTER_QA validate the data/signature and get PRINTER_QA to produce a similar signature based on the shared SoPEC_id key (i.e. the Translate function). SoPEC can then compare PRINTER_QA's signature with its own calculated signature (i.e. implement a Test function in software on the SoPEC), and if the signatures match, the data from INK_QA must be valid, and can therefore be trusted.

[0286] Once the data from INK_QA is known to be trusted, the amount of ink remaining can be checked, and the other ink licensing parameters such as OEM_Id, InkType, InkUsageLicense_Id can be checked for validity.

[0287] Strictly speaking, a nonce (R_{SOPEC}) is not needed all the time because M_A (containing the ink remaining) should be decrementing between authentications. However we do need one to retrieve the initial amount of ink and the other ink parameters (at power up). This is why taking a random number from the WatchDogTimer at the receipt of the first page is acceptable.

In summary, the SoPEC performs the non-authenticated write of ink remaining to the INK_QA chip, and then performs an authenticated read of the data via the PRINTER_QA as per the pseudocode above. If the value is authenticated, and the INK_QA ink-remaining value matches the expected value, the count was correctly decremented and the printing can continue.

[0288] Regarding broadcasting ink dot usage to all SoPECs in a multi-SoPEC system, in a multi-SoPEC system, each SoPEC attached to a printhead must broadcast its ink usage to all the SoPECs. In this way, each SoPEC will have its own version of the expected ink usage.

[0289] In the case of a man-in-the-middle attack, at worst the count in a given SoPEC is only its own count (i.e. all broadcasts are turned into 0 ink usage by the man-in-the-middle). We would also require the broadcast amount to be treated as an unsigned integer to prevent negative amounts from being substituted.

[0290] A single SoPEC performs the update of ink remaining to the INK_QA IC, and then all SoPECs perform an authenticated read of the data via the appropriate PRINTER_QA (the PRINTER_QA that contains their matching SoPEC_id key—remember that multiple SoPEC_id keys can be stored in a single PRINTER_QA). If the value is authenticated, and the INK_QA value matches the expected value, the count was correctly decremented and the printing can continue.

[0291] If any of the broadcasts are not received, or have been tampered with, the updated ink counts will not match. The only case this does not cater for is if each SoPEC is tricked (via a USB2 inter-SoPEC-comms man-in-the-middle attack) into a total that is the same, yet not the true total. Apart from the fact that this is not viable for general pages, at worst this is the maximum amount of ink printed by a single SoPEC. We don't care about protecting against this case.

[0292] Since a typical maximum is 4 printing SoPECs, it requires at most 4 authenticated reads. This should be completed within 0.5 seconds, well within the 1-2 seconds/page print time.

[0293] There must be a mapping of logical to physical since specific SoPECs are responsible for printing on particular physical parts of the page, and/or have particular devices attached to specific pins. The identification process is mostly solved by general USB2 enumeration.

[0294] Each slave SoPEC will need to verify the boot broadcast messages received over USB2, and only execute the code if the signatures are valid. Several levels of authorization may occur. However, at some stage, this common program code (broadcast to all of the slave SoPECs and signed by the appropriate asymmetric private key) can, among other things, set the slave SoPEC's id relating to the physical location. If there is only 1 slave, the id is easy to determine, but if there is more than 1 slave, the id must be determined in some fashion. For example, physical location/id determination may be:

- **[0295]** given by the physical USB2 port on the master
- **[0296]** related to the physical wiring up of the USB2 interconnects
- **[0297]** based on GPIO wiring. On other systems, a particular physical arrangement of SoPECs may exist such that each slave SoPEC will have a different set of connections on GPIOs. For example, one SoPEC maybe in charge of motor control, while another may be driving the LEDs etc. The unused GPIO pins (not necessarily the same on each SoPEC) can be set as inputs and then tied to 0 or 1. As long as the connection settings are mutually

exclusive, program code can determine which is which, and the id appropriately set.

[0298] This scheme of slave SoPEC_identification does not introduce a security breach. If an attacker rewires the pinouts to confuse identification, at best it will simply cause strange printouts (e.g. swapping of printout data) to occur, while at worst the Print Engine will simply not function.

[0299] The QA IC has its own internal memory, broken into the following conceptual regions:

[0300] RAM variables (3 Kbits=96 entries at 32-bits wide), used for scratch storage (e.g. HMAC-SHA1 processing).

[0301] Flash memory (8 Kbytes main block+128 bytes info block) used to hold the non-volatile authentication variables (including program keys etc), and program code. Only 4 KBytes+64 bytes is visible to the program addressing space due to shadowing. Shadowing is where half of each byte is used to validate and verify the other half, thus protecting against certain forms of physical and logical attacks. As a result, two bytes are read to obtain a single byte of data (this happens transparently).

[0302] The RAM region consists of 96×32-bit words required for the general functioning of the QA IC, but only during the operation of the chip. RAM is volatile memory: once power is removed, the values are lost. Note that in actual fact memory retains its value for some period of time after power-down, but cannot be considered to be available upon power-up. This has issues for security that are addressed in other sections of this document.

[0303] RAM is typically used for temporary storage of variables during chip operation. Short programs can also be stored and executed from the RAM.

[0304] RAM is addressed from 0 to 5F. Since RAM is in an unknown state upon a RESET (RstL), program code should not assume the contents to be **0**. Program code can, however, set the RAM to be a particular known state during execution of the reset command (guaranteed to be received before any other commands).

[0305] The flash memory region contains the non-volatile information in the QA IC. Flash memory retains its value after a RESET or if power is removed, and can be expected to be unchanged when the power is next turned on.

[0306] Byte **0** of main memory is the first byte of the program run for the command dispatcher. Note that the command dispatcher is always run with shadows enabled.

[0307] Bytes **0**-**7** of the information block flash memory is reserved as follows:

[0308] byte **0**-**3**=fuse. A value of 0x5555AAAA indicates that the fuse has been blown (think of a physical fuse whose wire is no longer intact).

[0309] bytes **4**-**7**=random number used to XOR all data for RAM and flash memory accesses

[0310] After power-on reset (when the fuse is blown) or upon receipt of a globalId Active command, the 32-bit data from bytes **4**-**7** in the information block of Flash memory is loaded into an internal ChipMask register. In Active Mode (the chip is executing program code), all data read from the flash and RAM is XORed with the ChipMask register, and all data written to the flash and RAM is XORed with the ChipMask register before being written out. This XORing happens completely transparently to the program code. Main flash memory byte **0** onward is the start of program code. Note that byte **0** onward needs to be valid after being XORed with the appropriate bytes of ChipMask.

[0311] Even though CPU access is in 8-bit and 32-bit quantities, the data is actually stored in flash a nybble-at-a-time. Each nybble write is written as a byte containing 4 sets of b/¬b pairs. Thus every byte write to flash is writing a nybble to real and shadow. A write mask allows the individual targetting of nybble-at-a-time writes.

[0312] The checking of flash vs shadow flash is automatically carried out each read (each byte contains both flash and shadow flash). If all 8 bits are 1, the byte is considered to be in its erased form (TSMC's flash memory has an erased state of all 1s), and returns 0 as the nybble. Otherwise, the value returned for the nybble depends on the size of the overall access and the setting of bit **0** of the 8-bit WriteMask.

[0313] All 8-bit accesses (i.e. instruction and program code fetches) are checked to ensure that each byte read from flash is 4 sets of b/¬b pairs. If the data is not of this form, the chip hangs until a new command is issued over the serial interface.

[0314] With 32-bit accesses (i.e. data used by program code), each byte read from flash is checked to ensure that it is 4 sets of b/¬b pairs. A setting of $\text{WriteMask}_0=0$ means that if the data is not valid, then the chip will hang until a new command is issued over the serial interface. A setting of $\text{WriteMask}_0=1$ means that each invalid nybble is replaced by the upper nybble of the WriteMask. This allows recovery after a write or erasure is interrupted by a power-down.

[0315] A high-level definition of a CPU capable of implementing the functionality required of an QA IC is as follows.

[0316] The pin connections to the QA IC are described in Table 2.

TABLE 2

Pin connections to QA IC

pin	direction	description
Vdd	In	Nominal voltage. If the voltage deviates from this by more than a fixed amount, the chip will RESET.
GND	In	
SClk	In	Serial clock
SDa	In/Out	Serial data

[0317] The system operating clock SysClk is different to SClk. SysClk is derived from an internal ring oscillator based on the process technology. In the FPGA implementation SysClk is obtained via a 5th pin.

[0318] The QA IC uses a 0.25 m CMOS Flash process for an area of 1 mm^2 yielding a 10 cent manufacturing cost in 2002. A breakdown of area is listed in Table 3.

TABLE 3

Breakdown of Area for QA IC

approximate area (mm^2)	description
0.49	8 KByte flash memory TSMC: SFC0008_08B9_HE (8K × 8-bits, erase page size = 512 bytes) Area = 724.688 m × 682.05 m.
0.08	3072 bits of static RAM
0.38	General logic
0.05	Analog circuitry
1	TOTAL (approximate)

Note that there is no specific test circuitry (scan chains or BIST) within the QA IC, so the total transistor count is as shown in Table 3.

[0319] The chip performs a RESET upon power-up. In addition, tamper detection and prevention circuitry in the chip will cause the chip to either RESET or erase Flash memory (depending on the attack detected) if an attack is detected.

[0320] The base operating system clock SysClk is generated internally from a ring oscillator (process dependant). Since the frequency varies with operating temperature and voltage, the clock is passed through a temperature-based clock filter before use. The frequency is built into the chip during manufacture, and cannot be changed. The frequency is in the range 7-14 MHz.

[0321] Manufacturing comments are not normally made when normally describing the architecture of a chip. However, in the case of the QA IC, the physical implementation of the chip is very much tied to the security of the key. Consequently a number of specialized circuits and components are necessary for implementation of the QA IC. They are listed here and described below:

[0322] Flash process

[0323] Internal randomized clock

[0324] Temperature based clock filter

[0325] Noise generator

[0326] Tamper Prevention and Detection circuitry

[0327] Protected memory with tamper detection

[0328] Boot-strap circuitry for loading program code

[0329] Data connections in polysilicon layers where possible

[0330] OverUnderPower Detection Unit

[0331] No scan-chains or BIST

[0332] The QA IC is implemented with a standard Flash manufacturing process. It is important that a Flash process be used to ensure that good endurance is achieved (parts of the Flash memory can be erased/written many times).

[0333] To prevent clock glitching and external clock-based attacks, the operating clock of the chip should be generated internally. This can be conveniently accomplished by an internal ring oscillator. The length of the ring depends on the process used for manufacturing the chip.

[0334] Due to process and temperature variations, the clock needs to be trimmed to bring it into a range usable for timing of Flash memory writes and erases.

[0335] The internal clock should also contain a small amount of randomization to prevent attacks where light emissions from switching events are captured, as described below. Finally, the generated clock must be passed through a temperature-based clock filter before being used by the rest of the chip.

[0336] The normal situation for FET implementation for the case of a CMOS inverter (which involves a pMOS transistor combined with an nMOS transistor) as shown in FIG. **18**.

[0337] During the transition, there is a small period of time where both the nMOS transistor and the pMOS transistor have an intermediate resistance. The resultant power-ground short circuit causes a temporary increase in the current, and in fact accounts for around 20% of current consumed by a CMOS device. A small amount of infrared light is emitted during the short circuit, and can be viewed through the silicon substrate (silicon is transparent to infrared light). A small amount of light is also emitted during the charging and discharging of the transistor gate capacitance and transmission line capacitance.

[0338] For circuitry that manipulates secret key information, such information must be kept hidden.

[0339] Fortunately, IBM's PICA system and LVP (laser voltage probe) both have a requirement for repeatability due to the fact that the photo emissions are extremely weak (one photon requires more than 10^5 switching events). PICA requires around 10^9 passes to build a picture of the optical waveform. Similarly the LVP requires multiple passes to ensure an adequate SNR.

[0340] Randomizing the clock stops repeatability (from the point of view of collecting information about the same position in time), and therefore reduces the possibility of this attack.

[0341] The QA IC circuitry is designed to operate within a specific clock speed range. Although the clock is generated by an internal ring oscillator, the speed varies with temperature and power. Since the user supplies the temperature and power, it is possible for an attacker to attempt to introduce race-conditions in the circuitry at specific times during processing. An example of this is where a low temperature causes a clock speed higher than the circuitry is designed for, and this may prevent an XOR from working properly, and of the two inputs, the first may always be returned. The lesson to be learned from this is that the input power and operating temperature cannot be trusted.

[0342] Since the chip contains a specific power filter, we must also filter the clock. This can be achieved with a temperature sensor that allows the clock pulses through only when the temperature range is such that the chip can function correctly.

[0343] The filtered clock signal would be further divided internally as required.

[0344] Each QA IC should contain a noise generator that generates continuous circuit noise. The noise will interfere with other electromagnetic emissions from the chip's regular activities and add noise to the I_{dd} signal. Placement of the noise generator is not an issue on an QA IC due to the length of the emission wavelengths.

[0345] The noise generator is used to generate electronic noise, multiple state changes each clock cycle, and as a source of pseudo-random bits for the Tamper Prevention and Detection circuitry.

[0346] A simple implementation of a noise generator is a 64-bit maximal period LFSR seeded with a non-zero number.

[0347] A set of circuits is required to test for and prevent physical attacks on the QA IC. However what is actually detected as an attack may not be an intentional physical attack. It is therefore important to distinguish between these two types of attacks in an QA IC:

[0348] where you can be certain that a physical attack has occurred.

[0349] where you cannot be certain that a physical attack has occurred.

[0350] The two types of detection differ in what is performed as a result of the detection. In the first case, where the circuitry can be certain that a true physical attack has occurred, erasure of flash memory key information is a sensible action. In the second case, where the circuitry cannot be sure if an attack has occurred, there is still certainly something wrong. Action must be taken, but the action should not be the erasure of secret key information. A suitable action to take in the second case is a chip RESET. If what was detected was an attack that has permanently damaged the chip, the same conditions will occur next time and the chip will RESET again. If, on the other hand, what was detected was part of the normal operating environment of the chip, a RESET will not harm the key.

[0351] A good example of an event that circuitry cannot have knowledge about, is a power glitch. The glitch may be an intentional attack, attempting to reveal information about the key. It may, however, be the result of a faulty connection, or simply the start of a power-down sequence. It is therefore best to only RESET the chip, and not erase the key. If the chip was powering down, nothing is lost. If the System is faulty, repeated RESETs will cause the consumer to get the System repaired. In both cases the consumable is still intact.

[0352] A good example of an event that circuitry can have knowledge about, is the cutting of a data line within the chip. If this attack is somehow detected, it could only be a result of a faulty chip (manufacturing defect) or an attack. In either case, the erasure of the secret information is a sensible step to take.

[0353] Consequently each QA IC should have 2 Tamper Detection Lines—one for definite attacks, and one for possible attacks. Connected to these Tamper Detection Lines would be a number of Tamper Detection test units, each testing for different forms of tampering. In addition, we want to ensure that the Tamper Detection Lines and Circuits themselves cannot also be tampered with.

[0354] At one end of the Tamper Detection Line is a source of pseudo-random bits (clocking at high speed compared to the general operating circuitry). The Noise Generator circuit described above is an adequate source. The generated bits pass through two different paths—one carries the original data, and the other carries the inverse of the data. The wires carrying these bits are in the layer above the general chip circuitry (for example, the memory, the key manipulation circuitry etc.). The wires must also cover the random bit generator. The bits are recombined at a number of places via an XOR gate. If the bits are different (they should be), a 1 is output, and used by the particular unit (for example, each output bit from a memory read should be ANDed with this bit value). The lines finally come together at the Flash memory Erase circuit, where a complete erasure is triggered by a 0 from the XOR. Attached to the line is a number of triggers, each detecting a physical attack on the chip. Each trigger has an oversize nMOS transistor attached to GND. The Tamper Detection Line physically goes through this nMOS transistor. If the test fails, the trigger causes the Tamper Detect Line to become 0. The XOR test will therefore fail on either this clock cycle or the next one (on average), thus RESETing or erasing the chip.

[0355] FIG. **14** illustrates the basic principle of a Tamper Detection Line in terms of tests and the XOR connected to either the Erase or RESET circuitry.

[0356] The Tamper Detection Line must go through the drain of an output transistor for each test, as illustrated by FIG. **15**.

[0357] It is not possible to break the Tamper Detect Line since this would stop the flow of 1s and 0s from the random source. The XOR tests would therefore fail. As the Tamper Detect Line physically passes through each test, it is not possible to eliminate any particular test without breaking the Tamper Detect Line.

[0358] It is important that the XORs take values from a variety of places along the Tamper Detect Lines in order to reduce the chances of an attack. FIG. **16** illustrates the taking of multiple XORs from the Tamper Detect Line to be used in the different parts of the chip. Each of these XORs can be considered to be generating a ChipOK bit that can be used within each unit or sub-unit.

[0359] A typical usage would be to have an OK bit in each unit that is ANDed with a given ChipOK bit each cycle. The OK bit is loaded with 1 on a RESET. If OK is 0, that unit will fail until the next RESET. If the Tamper Detect Line is functioning correctly, the chip will either RESET or erase all key information. If the RESET or erase circuitry has been destroyed, then this unit will not function, thus thwarting an attacker.

[0360] The destination of the RESET and Erase line and associated circuitry is very context sensitive. It needs to be protected in much the same way as the individual tamper tests. There is no point generating a RESET pulse if the attacker can simply cut the wire leading to the RESET circuitry. The actual implementation will depend very much on what is to be cleared at RESET, and how those items are cleared.

[0361] Finally, FIG. **17** shows how the Tamper Lines cover the noise generator circuitry of the chip. The generator and NOT gate are on one level, while the Tamper Detect Lines run on a level above the generator.

[0362] It is not enough to simply store secret information or program code in flash memory. The Flash memory and RAM must be protected from an attacker who would attempt to modify (or set) a particular bit of program code or key information. The mechanism used must conform to being used in the Tamper Detection Circuitry (described above).

[0363] The first part of the solution is to ensure that the Tamper Detection Line passes directly above each flash or RAM bit. This ensures that an attacker cannot probe the contents of flash or RAM. A breach of the covering wire is a break in the Tamper Detection Line. The breach causes the Erase signal to be set, thus deleting any contents of the memory. The high frequency noise on the Tamper Detection Line also obscures passive observation.

[0364] The second part of the solution for flash is to always store the data with its inverse. In each byte, 4 bits contains the data, and 4 bits (the shadow) contains the inverse of the data. If both are 0, this is a valid erase state, and the value is 0. Otherwise, the memory is only valid if the 4 bits of shadow are the inverse of the main 4 bits. The reasoning is that it is possible to add electrons to flash via a FIB, but not take electrons away. If it is possible to change a 0 to 1 for example, it is not possible to do the same to its inverse, and therefore regardless of the sense of flash, an attack can be detected.

[0365] The second part of the solution for RAM is to use a parity bit. The data part of the register can be checked against the parity bit (which will not match after an attack).

[0366] The bits coming from Flash and RAM can therefore be validated by a number of test units (one per bit) connected to the common Tamper Detection Line. The Tamper Detection circuitry would be the first circuitry the data passes through (thus stopping an attacker from cutting the data lines).

[0367] In addition, the data and program code should be stored in different locations for each chip, so an attacker does not know where to launch an attack. Finally, XORing the data coming in and going to Flash with a random number that varies for each chip means that the attacker cannot learn anything about the key by setting or clearing an individual bit that has a probability of being the key (the inverse of the key must also be stored somewhere in flash).

[0368] Finally, each time the chip is called, every flash location is read before performing any program code. This allows the flash tamper detection to be activated in a common spot instead of when the data is actually used or program code executed. This reduces the ability of an attacker to know exactly what was written to.

[0369] Program code should be kept in protected flash instead of ROM, since ROM is subject to being altered in a non-testable way. A boot-strap mechanism is therefore required to load the program code into flash memory (flash memory is in an indeterminate state after manufacture).

[0370] The boot-strap circuitry must not be in a ROM—a small state-machine suffices. Otherwise the boot code could be trivially modified in an undetectable way.

[0371] The boot-strap circuitry must erase all flash memory, check to ensure the erasure worked, and then load the program code.

[0372] The program code should only be executed once the flash program memory has been validated via Program Mode.

[0373] Once the final program has been loaded, a fuse can be blown to prevent further programming of the chip.

[0374] Wherever possible, the connections along which the key or secret data flows, should be made in the polysilicon layers. Where necessary, they can be in metal **1**, but must never be in the top metal layer (containing the Tamper Detection Lines).

[0375] Each QA IC requires an OverUnder Power Detection Unit (PDU) to prevent Power Supply Attacks. A PDU detects power glitches and tests the power level against a Voltage Reference to ensure it is within a certain tolerance. The Unit contains a single Voltage Reference and two comparators. The PDU would be connected into the RESET Tamper Detection Line, thus causing a RESET when triggered.

[0376] A side effect of the PDU is that as the voltage drops during a power-down, a RESET is triggered, thus erasing any work registers.

[0377] Test hardware on an QA IC could very easily introduce vulnerabilities. In addition, due to the small size of the QA IC logic, test hardware such as scan paths and BIST units could in fact take a sizeable chunk of the final chip, lowering yield and causing a situation where an error in the test hardware causes the chip to be unusable. As a result, the QA IC should not contain any BIST or scan paths. Instead, the program memory must first be validated via the Program Mode mechanism, and then a series of program tests run to verify the remaining parts of the chip.

[0378] FIG. **19** shows a high level block diagram of the QA IC. Note that the tamper prevention and detection circuitry is not shown.

[0379] FIG. **20** shows a block diagram of the Analogue Unit. Blocks shown in yellow provide additional protection against physical and electrical attack and, depending on the level of security required, may optionally be implemented.

[0380] The operating clock of the chip (SysClk) is generated by an internal ring oscillator whose frequency can be trimmed to reduce the variation from 4:1 (due to process and temperature) down to 2:1 (temperature variations only) in order to satisfy the timing requirements of the Flash memory.

[0381] The length of the ring depends on the process used for manufacturing the chip. A nominal operating frequency range of 10 MHz is sufficient. This clock should contain a small amount of randomization to prevent attacks where light emissions from switching events are captured.

[0382] Note that this is different to the input SClk which is the serial clock for external communication.

[0383] The ring oscillator is covered by both Tamper Detection and Prevention lines so that if an attacker attempts to tamper with the unit, the chip will either RESET or erase all secret information.

[0384] The voltage reference block maintains an output which is substantially independant of process, supply voltage and temperature. It provides a reference voltage which is used by the PDU and a reference current to stabilise the ring oscillator. It may also be used as part of the temperature based clock filter.

[0385] The Under Voltage Detection Unit provides the signal PwrFailing which, if asserted, indicates that the power supply may be turning off. This signal is used to rapidly terminate any Flash write that may be in progress to avoid accidentally writing to an indeterminate memory location. Note that the PDU triggers the RESET Tamper Detection Line only. It does not trigger the Erase Tamper Detection Line.

[0386] The PDU can be implemented with regular CMOS, since the key does not pass through this unit. It does not have to be implemented with non-flashing CMOS.

[0387] The PDU is covered by both Tamper Detection and Prevention lines so that if an attacker attempts to tamper with the unit, the chip will either RESET or erase all secret information.

[0388] The Power-on Reset unit (POR) detects a power-on condition and generates the PORstL signal that is fed to all the validation units, including the two inside the Tamper Detect Unit (TDU).

[0389] All other logic is connected to RstL, which is the PORstL gated by the VAL unit attached to the Reset tamper detection lines within the TDU. Therefore, if the Reset tamper line is asserted, the validation will drive RstL low, and can only be cleared by a power-down. If the tamper line is not asserted, then RstL=PORstL.

[0390] The TDU contains a second VAL unit attached to the Erase tamper detection lines within the TDU. It produces a TamperEraseOK signal that is output to the MIU (1=the tamper lines are all OK, 0=force an erasure of Flash).

[0391] The Noise Generator (NG) is based on a 64-bit maximal period LFSR loaded with a set non-zero bit pattern on RESET.

[0392] The NG must be protected by both Tamper Detection and Prevention lines so that if an attacker attempts to tamper with the unit, the chip will either RESET or erase all secret information.

[0393] In addition, the bits in the LFSR must be validated to ensure they have not been tampered with (i.e. a parity check). If the parity check fails, the Erase Tamper Detection Line is triggered.

[0394] Finally, all 64 bits of the NG are ORed into a single bit. If this bit is 0, the Erase Tamper Detection Line is triggered. This is because 0 is an invalid state for an LFSR.

[0395] The 8-bit Trim register within the Trim Unit has a reset value of 0x00 (to enable the flash reads to succeed even in the fastest process corners), and is written to either by the PMU during Trim Mode or by the CPU in Active Mode. Note that the CPU is only able to write once to the Trim register between power-on-reset due to the TrimDone flag which provides overloading of LocalIdWE.

[0396] The reset value of Trim (0) means that the chip has a nominal frequency of 2.7 MHz-10 MHz. The upper of the range is when we cannot trim it lower than this (or we could allow some spread on the acceptable trimmed frequency but this will reduce our tolerance to ageing, voltage and temperature which is the range 7 MHz to 14 MHz). The 2.7 MHz value is determined by a chip whose oscillator runs at 10 MHz when the trim register is set to its maximum value, so then it must run at 2.7 MHz when trim=0. This is based on the non-linear frequency-current characteristic of the oscillator. Chips found outside of these limits will be rejected.

[0397] The frequency of the ring oscillator is measured by counting cycles, in the PMU, over the byte period of the serial interface. Note that the PMU counts using 12-bits, saturates at 0xFFF, and returns the cycle count divided by 2 as an 8-bit value. This means that multiple measure-read-trim cycles may be necessary to resolve any ambiguity. In any case, multiple cycles are necessary to test the correctness of the trim circuitry during manufacture test.

[0398] The frequency of the serial clock, SClk, and therefore the byte period will be accurately controlled during the measurement. The cycle count (Fmeas) at the end of the period is read over the serial bus and the Trim register updated (Trimval) from its power on default (POD) value. The steps are shown in FIG. **21**. Multiple measure—read—trim cycles are possible to improve the accuracy of the trim procedure.

[0399] A single byte for both Fmeas and Trimval provide sufficient accuracy for measurement and trimming of the frequency. If the bus operates at 400 kHz, a byte (8 bits) can be sent in 20 s. By dividing the maximum oscillator frequency, expected to be 20 MHz, by 2 results in a cycle count of 200 and 50 for the minimum frequency of 5 MHz resulting in a worst case accuracy of 2%. FIG. **22** shows a block diagram of the Trim Unit.

[0400] The 8-bit Trim value is used in the analog Trim Block to adjust the frequency of the ring oscillator by controlling its bias current. The two lsbs are used as a voltage trim, and the 6 msbs are used as a frequency trim. The analog Trim Clock circuit also contains a Temperature filter.

[0401] The QA IC acts as a slave device, accepting serial data from an external master via the IO Unit (IOU). Although the IOU actually transmits data over a 1-bit line, the data is always transmitted and received in 1-byte chunks.

[0402] The IOU receives commands from the master to place it in a specific operating mode, which is one of:

[0403] Idle Mode: is the startup mode for the IOU if the fuse has not yet been blown. Idle Mode is the mode where the QA IC is waiting for the next command from the master. Input signals from the CPU are ignored.

[0404] Program Mode: is where the QA IC erases all currently stored data in the Flash memory (program and secret key information) and then allows new data to be written to the Flash. The IOU stays in Program Mode until told to enter another mode.

[0405] Active Mode: is the startup mode for the IOU if the fuse has been blown (the program is safe to run). Active Mode is where the QA IC allows the program code to be executed to process the master's specific command. The IOU returns to Idle Mode automatically when the command has been processed, or if the time taken between consuming input bytes (while the master is writing the data) or generating output bytes (while the master is reading the results) is too great.

[0406] Trim Mode: is where the QA IC allows the generation and setting of a trim value to be used on the internal ring oscillator clock value. This must be done for safety reasons before a program can be stored in the Flash memory.

[0407] The Central Processing Unit (CPU) block provides the majority of the circuitry of the 4-bit microprocessor. FIG. **23** shows a high level view of the block.

[0408] The Memory Interface Unit (MIU) provides the interface to flash and RAM. The MIU contains a Program Mode Unit that allows flash memory to be loaded via the IOU, a Memory Request Unit that maps 8-bit and 32-bit requests into multiple byte based requests, and a Memory Access Unit that generates read/write strobes for individual accesses to the memory. FIG. **24** shows a high level view of the MIU block.

[0409] The Memory Components block isolates the memory implementation from the rest of the QA IC.

[0410] The entire contents of the Memory Components block must be protected from tampering. Therefore the logic must be covered by both Tamper Detection Lines. This is to ensure that program code, keys, and intermediate data values cannot be changed by an attacker. The 8-bit wide RAM also needs to be parity-checked.

[0411] FIG. **25** shows a high level view of the Memory Components block. It consists of 8 KBytes of flash memory and 3072 bits of parity checked RAM.

[0412] The RAM block is shown here as a simple 96×32-bit RAM (plus parity included for verification). The parity bit is generated during the write.

[0413] The RAM is in an unknown state after RESET, so program code cannot rely on RAM being 0 at startup.

[0414] The initial version of the ASIC has the RAM implemented by Artisan component RA1 SH (96×32-bit RAM without parity). Note that the RAMOutEn port is active low i.e. when 0, the RAM is enabled, and when 1, the RAM is disabled.

[0415] A single Flash memory block is used to hold all non-volatile data. This includes program code and variables. The Flash memory block is implemented by TSMC component SFC0008_08B9_HE [4], which has the following characteristics:

[0416] 8K×8-bit main memory, plus 128×8-bit information memory

[0417] 512 byte page erase

[0418] Endurance of 20,000 cycles (min)

[0419] Greater than 100 years data retention at room temperature

[0420] Access time: 20 ns (max)

[0421] Byte write time: 20 s (min)

[0422] Page erase time: 20 ms (min)

[0423] Device erase time: 200 ms (min)

[0424] Area of 0.494 mm^2 (724.66 m×682.05 m)

[0425] The FlashCtrl line are the various inputs on the SFC0008_08B9_HE required to read and write bytes, erase pages and erase the device. A total of 9 bits are required.

[0426] Flash values are unchanged by a RESET. After manufacture, the Flash contents must be considered to be garbage. After an erasure, the Flash contents in the SFC0008_08B9_HE is all 1s.

[0427] The two VAL units are validation units connected to the Tamper Prevention and Detection circuitry, each with an OK bit. The OK bit is set to 1 on PORstL, and ORed with the

ChipOK values from both Tamper Detection Lines each cycle. The OK bit is ANDed with each data bit that passes through the unit.

[0428] In the case of VAL_1, the effective byte output from the flash will always be 0 if the chip has been tampered with. This will cause shadow tests to fail, program code will not execute, and the chip will hang.

[0429] In the case of VAL_2, the effective byte from RAM will always be 0 if the chip has been tampered with, thus resulting in no temporary storage for use by an attacker.

[0430] It would be appreciated by a person skilled in the art that numerous variations and/or modifications may be made to the present invention as shown in the specific embodiment without departing from the spirit or scope of the invention as broadly described. The present embodiments are, therefore, to be considered in all respects to be illustrative and not restrictive.

1. An integrated circuit configured to:

run a boot program that verifies programs before said programs can be loaded onto, or run by, the integrated circuit by verifying whether said programs are signed with a boot key;

verify, with the boot program, a developmental boot program signed with the boot key which verifies developmental programs before said developmental programs can be loaded onto, or run by, the integrated circuit by verifying whether the integrated circuit has a predetermined integrated circuit identifier; and

load the verified developmental boot program and run the loaded developmental booth program thereby enabling loading or running of said developmental programs on the integrated circuit if the integrated circuit has the predetermined integrated circuit identifier, and

programmed with program code configured to:

receive encrypted software data,

decrypt the software data; and

validate the software data,

wherein the decrypted software is executed only when the validation is successful.

2. An integrated circuit according to claim **1**, wherein the encryption function is RSA.

3. An integrated circuit according to claim **1**, wherein the boot program contains a plurality of keys, and one of the keys is selected for use in decrypting the software data, the key being selected in accordance with a selection criterion.

4. An integrated circuit according to claim **3**, wherein the selection criterion is time-based, a particular one of the keys being selected depending on the time the selection is made.

5. An integrated circuit according to claim **3**, wherein the selection criteria relates to a physical arrangement or configuration of the integrated circuit.

6. An integrated circuit according to claim **5**, wherein the physical arrangement or configuration includes one or more of the following:

one or more pads wired to a reference voltage or to ground;

one or more fuses, one or more of which has been blown; or

the contents of non-volatile memory.

* * * * *