CHAPTER 1 CSL MBIST

TABLE 1.1 Chapter Overview

1.1 CSL MBIST Overview

MBIST - Memory Built-in-Self Test is the technique of designing additional hardware and software features into memories to allow them to perform self-testing using their own circuits, thereby reducing dependence on an external automated test equipment. It typically consists of test circuits that apply, read, and compare test patterns designed to expose defects in the memory device.

1.2 CSL MBIST Concept

1.2.1 MBIST window tests

1.2.1.1 MBIST master controller

TABLE 1.2

0	0	0	0	0
0	1	1	1	0
0	1	0	1	0

TABLE 1.2

0	0	0	0	0
0	1	1	1	0
0	0	0	0	0

TABLE 1.3 march0

1	1	1	1	1
1	1	1	1	1
1	1	0	1	1
1	1	1	1	1
1	1	1	1	1

TABLE 1.4 march1

0	0	0	0	0
0	0	0	0	0
0	0	1	0	0
0	0	0	0	0
0	0	0	0	0

TABLE 1.5

1	1	1	1	1
1	0	0	0	1
1	0	1	0	1
1	0	0	0	1
1	1	1	1	1

1.2.1.2 MBIST client controller pattern rom

MemBist
what is being tested

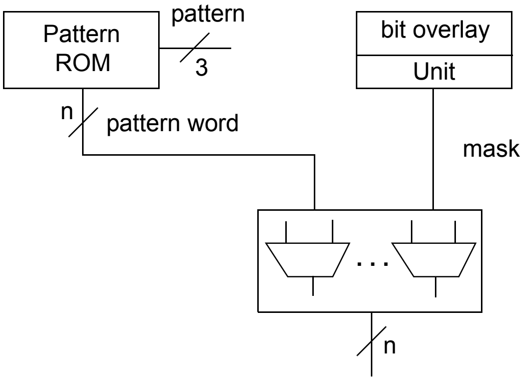
1. bad address decoder
2. stuck @ faults

3. neighbor cell coupling

4.

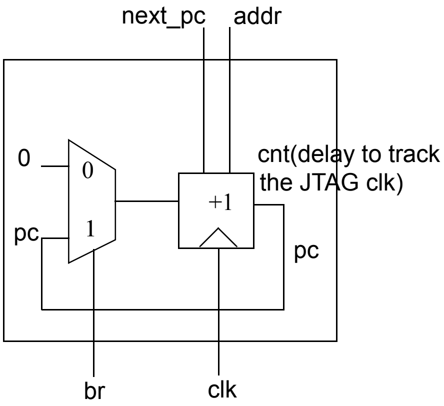
Mem Bist
pattern generator = background
overlay bit generator
The overlay bit generator increments the x counter (which may wrap)
The x counter determines tracks where to insert 0/1 bit into the word generated by the pattern generator.
The march type specifies whether the overlay bit unit is activated meaning that the overlay bit is inserted into the pattern word.

FIGURE 1.1 Mem bist



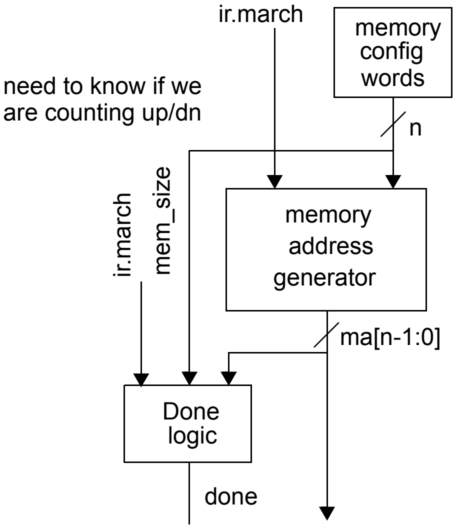
1.2.1.3 MBIST master controller PC

FIGURE 1.2 MBIST PC unit



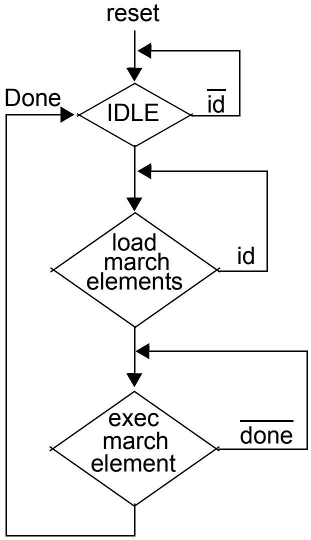
1.2.1.4 MBIST client controller address generator

FIGURE 1.3 MBIST client controller address generator



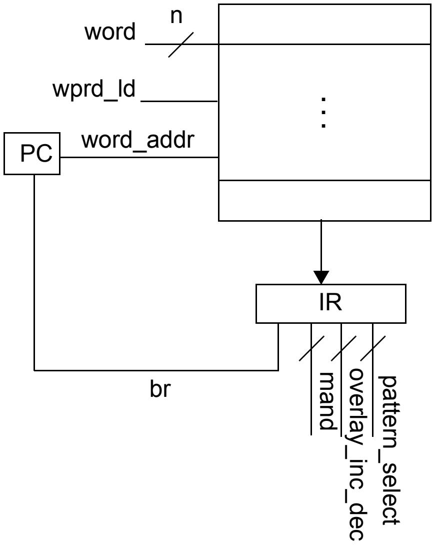
1.2.1.5 MBIST client controller FSM

FIGURE 1.4 MBIST client controller FSM



1.2.1.6 MBIST microengine instruction memory

FIGURE 1.5 MBIST microengine instruction memory & PC



1.2.1.7 MBIST chip hierarchy

FIGURE 1.6

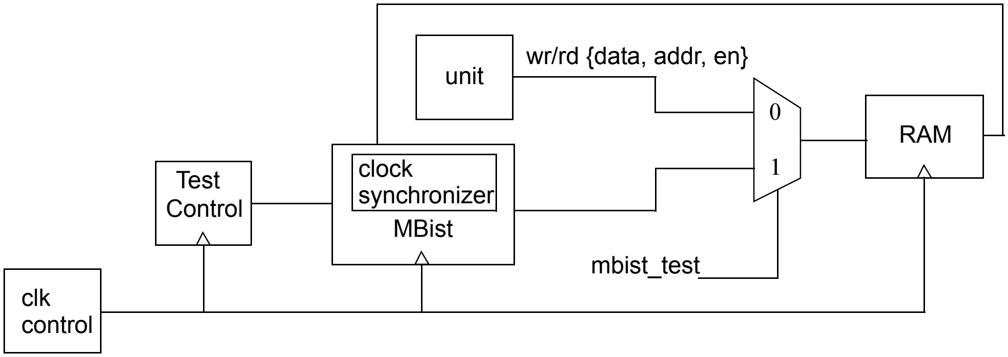


TABLE 1.6

BIST	test	for faults
BISR	redundancy	replace bad SRAM columns

1.2.1.8 MBIST master controller microengine

FIGURE 1.7 MBIST master controller microengine

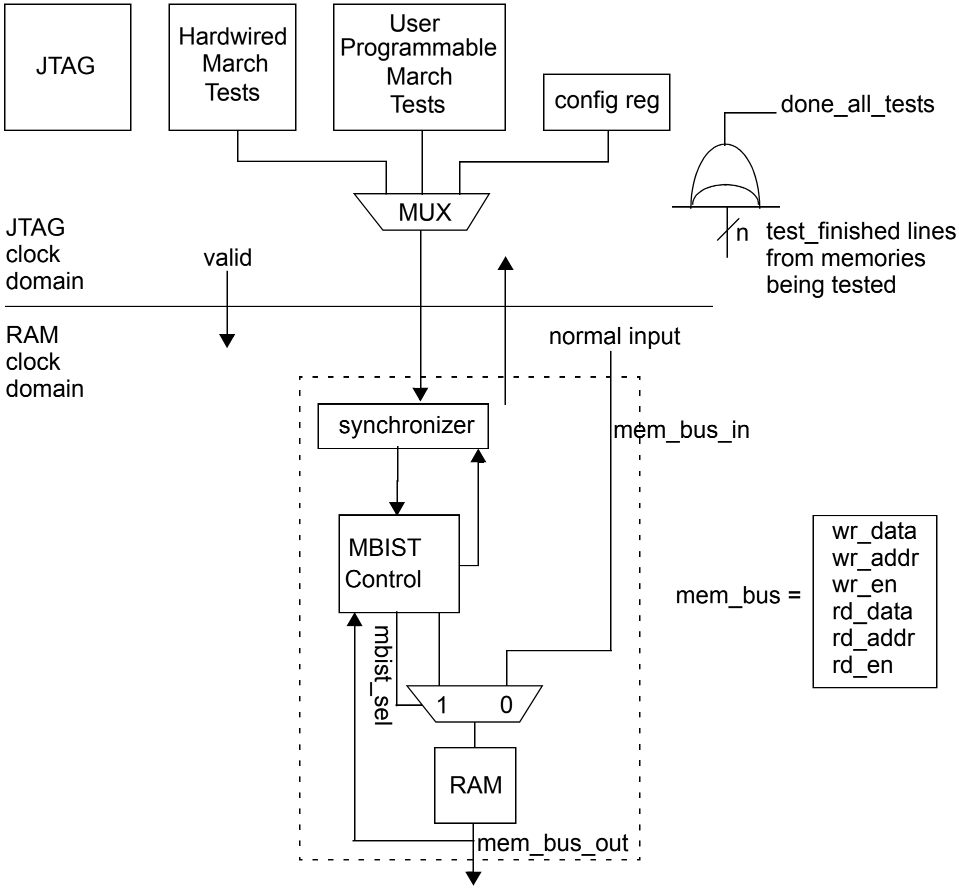
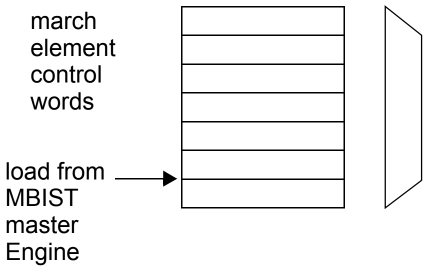


FIGURE 1.8 MBIST



1.2.1.9 MBIST test patterns

MBIST Data Patterns

0 ----------------- 0 1 0 ----------------- 0
1 ----------------- 1 0 1 ----------------- 1
0/1 moves through word
Back grounds

TABLE 1.7 Checker board

0	1	0	1	0	1
1	0	1	0	1	0
0	1	0	1	0	1
1	0	1	0	1	0
0	1	0	1	0	1
1	0	1	0	1	0

TABLE 1.8 Inverse checker board

1	0	1	0	1	0
0	1	0	1	0	1
1	0	1	0	1	0
0	1	0	1	0	1
1	0	1	0	1	0
0	1	0	1	0	1

0 -------------------------- 0 all 0’s
1 -------------------------- 1 all 1’s

1.2.1.10 MBIST test patterns

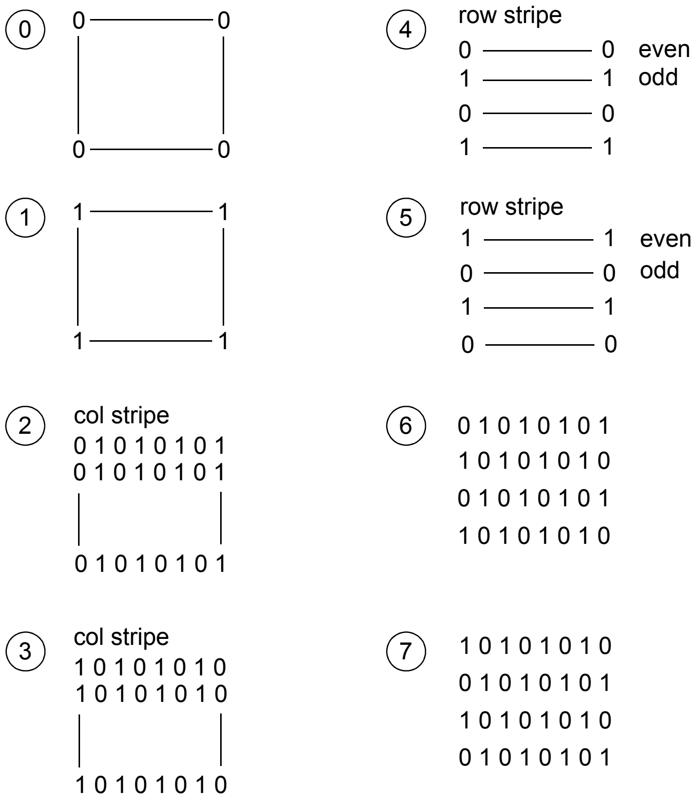
MBIST control word

Background

TABLE 1.9

RW	
0	rd
1	wr

FIGURE 1.9 Data Back ground



1.2.1.11 MBIST test tables

TABLE 1.10

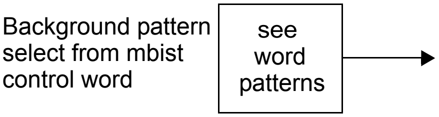
March LA	single cell faults	AF, SAF, TF, SOF, DRF, RDF
	inter-word linked multicell	CFst, CFin, CFid, CFds
	inter-word linked multicell faults	TF & CFin, CFid & CFid, CFid & CFin, CFst & CFin, CFds & CFds, CFds & CFin
Full Move	single cell faults	AF, SAF, TF, SOF, DRF, RDF
	inter-word unlinked multicell faults	CFst, CFin, CFid, CFds
	inter-word linked multicell faults	CFid & CFin
	address delay logic	
	read delay logic	
DBOS for rCFst	intra-word multicell faults	rCFst, rCFin
	intra-word single cell faults	AF, SAF, TF, SOF
2PF 2 ar S	Do RAM allow simultaneous rd/wr access to the same address	
nPFnavs		
March LR	single cell faults	AF, SAF, TF, SOF, DRF, RDF
	inter-word unlinked multicell faults	TF & CFin, CFin & CFid, CFin & CFst, CFin & CFds, CFid & CFid, CFds & Cfds

Checker Board	DBStress	SAF	AF	TF	CF
Column stripe	DBSTress				
Row stripe	DBStress				

Reliable
Scalable

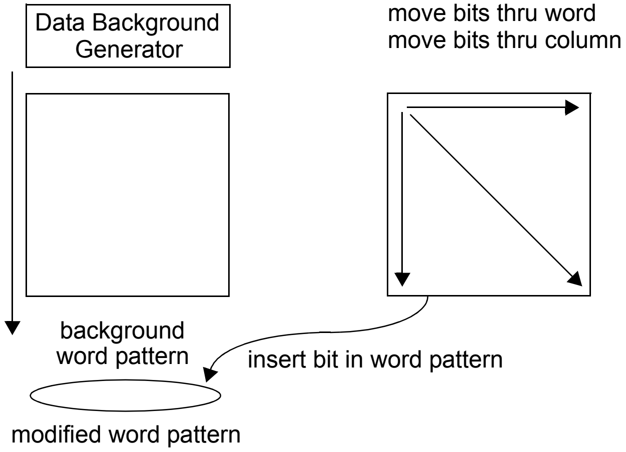
MBIST

FIGURE 1.10 Background ROM



1.2.1.12 MBIST test patterns

FIGURE 1.11



1.2.1.13 MBIST test tables

Relationship between functional and reduced functional faults

TABLE 1.11 Fault table

Fault category	Fault type
SAF	Cell stuck at 0
SAF	Cell stuck at 1
SAF	Driver stuck at 0 or 1
SAF	Read line pulled down
SAF	Read line pulled up
SAF	Write line pulled down
SAF	Write line pulled up
SAF	Data line stuck
SAF	Open data line
SAF	Open data line
CF	Shorts between data lines
CF	Croostalk between data lines
AF	Address line pulled down
AF	Address line pulled up
AF	Open in address lines
AF	Shorts between address lines
AF	Open decoder
AF	Wrong access
AF	Multiple access
TF	Cell can be set to 0 But not to 1 (or vice-versa)
NPSF	Pattern sensitive iteration between cells

1.2.1.14 MBIST memory faults

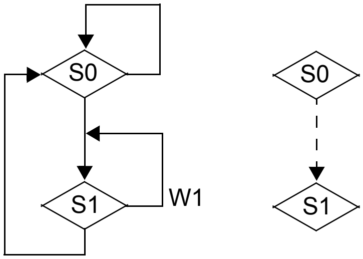
Coupling Fault
2-coupling faults

- Idempotent coupling fault
- Inversion coupling fault

K-coupling fault

1.2.1.15 MBIST fault pattern state machine

FIGURE 1.12 Neighborhood Pattern Sensitive fault



Notation of March Tests
A march element is a finite sequence of operations applied to every cell.

FIGURE 1.13

⇑ Increasing address order

⇓ Decreasing address order

A march test is a finite sequence of march elements.

FIGURE 1.14 Stuck At Fault (SAF)

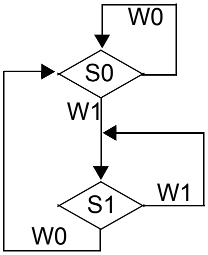


FIGURE 1.15 State diagram of a good cell

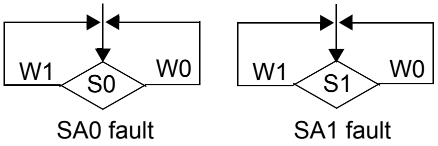
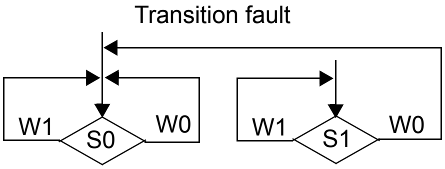


FIGURE 1.16 Transition Fault (TF)



1.2.1.16 March tests

March Test Notation
^ = increasing address 0..n
v = decreasing address 0..n
| = don't care, 0..n or n..0
w = w0 or w1 write data
r = r0 or r1 read data and verify
del = delay

Popular March Tests

-MATS+ (Abadir & Reghbati, 1983)

```
|(w0);^(r0,w1);v(r1,w0);
```

PSEUDO CODE

```
m = memory;
not_run=10;
passed=1;
failed=0;
status=not_run;
for i = 0 to n-1
   m(i) = 0
for i = 0 to n-1
   if m(i) = 0
     then m(i) = 1
     else return fail
for i = n-1 to 0
   if m(i) = 1
     then m(i) = 0
     else return fail
return pass
```

ASM CODE

```
//loop0
    limit=0;
    mov r2 limit;
  loop0:
    mov r1 r0; //r0=const0
    mov mem[r1] r0;
    sub r3 r2 r1; //r3=r2-r1
    br n0 r3 loop0; //branch to loop0 if r3!=0
//loop1
      limit=0;
      mov r2 limit;
    loop1:
      mov r1 r0;
      mov mem[r1] r0;
      sub r3 r2 r1;
      br n0 r3 loop1;
//loop1
      limit=0;
      mov r2 limit;
    loop1:
      mov r1 r0;
      mov mem[r1] r0;
      sub r3 r2 r1;
      br n0 r3 loop1;
//loop1
      limit=0;
      mov r2 limit;
    loop1:
      mov r1 r0;
      mov mem[r1] r0;
      sub r3 r2 r1;
      br n0 r3 loop1;
//loop1
      limit=0;
      mov r2 limit;
    loop1:
      mov r1 r0;
      mov mem[r1] r0;
      sub r3 r2 r1;
      br n0 r3 loop1;
//loop2
      limit=n-1;
      mov r2 limit;
    loop2:
      mov r1 r0;
      mov mem[r1] r0;
      sub r3 r2 r1;
       br n0 r3 loop2;
```

C Code

```
//m is the SRAM memory array with the dimension n
#define DIMENSION n
unsigned char m[DIMENSION];
bool mats_plus(void)
{
  int i;
  for(i = 0; i < DIMENSION; i++)
```

```
    m[i] = 0;
  for(i = 0; i < DIMENSION; i++){
    if(m[i] == 0)
      m[i] = 1;
    else
      return false;
  }
  for(i = DIMENSION-1; i >= 0; i--){
    if(m[i] == 1)
      m[i] = 0;
    else
      return false;
  }
  return true;
}
```

-MATS++ (Van de Goor, 1991)

```
|(w0);^(r0,w1);v(r1,w0,r0);
```

PSEUDO CODE

```
for i = 0 to n-1
  do m(i) = 0
for i = 0 to n-1
  do if m(i) = 0
     then m(i) = 1
     else return fail
for i = n-1 to 0
  do if m(i) = 1
     then m(i) = 0
     else return fail
     if m(i) = 0
     then
     else return fail
return pass
```

C CODE

```
//m is the SRAM memory array with the dimension n
#define DIMENSION n
unsigned char m[DIMENSION];
bool mats_plus_plus(void)
{
  int i;
  for(i = 0; i < DIMENSION; i++)
    m[i] = 0;
  for(i = 0; i < DIMENSION; i++){
    if(m[i] == 0)
      m[i] = 1;
    else
      return false;
  }
  for(i = DIMENSION-1; i >= 0; i--){
    if(m[i] == 1)
      m[i] = 0;
    else
      return false;
    if(m[i] == 0)
      ;
    else
      return false;
  }
  return true;
}
```

-MARCH C- (Van de Goor, 1991)

```
|(w0);^(r0,w1);^(r1,w0);v(r0,w1);v(r1,w0);|(r0)
```

PSEUDO CODE

```
for i = 0 to n-1
  do m(i) = 0
for i = 0 to n-1
  do if m(i) = 0
     then m(i) = 1
     else return fail
for i = 0 to n-1
  do if m(i) = 1
     then m(i) = 0
     else return fail
for i = n-1 to 0
  do if m(i) = 0
     then m(i) = 1
     else return fail
for i = n-1 to 0
  do if m(i) = 1
     then m(i) = 0
     else return fail
for i = 0 to n-1
  do if m(i) = 0
     then
     else return fail
return pass
```

C CODE

```
//m is the SRAM memory array with the dimension n
#define DIMENSION n
unsigned char m[DIMENSION];
bool march_c_minus(void)
{
  int i;
  for(i = 0; i < DIMENSION; i++)
    m[i] = 0;
  for(i = 0; i < DIMENSION; i++){
    if(m[i] == 0)
      m[i] = 1;
    else
```

```
      return false;
  }
  for(i = 0; i < DIMENSION; i++){
    if(m[i] == 1)
      m[i] = 0;
    else
      return false;
  }
  for(i = DIMENSION-1; i >= 0; i--){
    if(m[i] == 0)
      m[i] = 1;
    else
      return false;
  }
  for(i = DIMENSION; i >= 0; i--){
    if(m[i] == 1)
      m[i] = 0;
    else
      return false;
  }
  for(i = 0; i < DIMENSION; i++){
    if(m[i] == 0)
      ;
    else
      return false;
  }
  return true;
}
```

-MARCH B (Suk, 1981)

```
|(w0);^(r0,w1,r1,w0,r0,w1);^(r1,w0,w1);
v(r1,w0,w1,w0);v(r0,w1,w0)
```

PSEUDO CODE

```
for i = 0 to n-1
  do m(i) = 0
for i = 0 to n-1
  do if m(i) = 0
     then m(i) = 1
     else return fail
     if m(i) = 1
     then m(i) = 0
     else return fail
     if m(i) = 0
     then m(i) = 1
     else return fail
for i = 0 to n-1
  do if m(i) = 1
     then m(i) = 0
          m(i) = 1
     else return fail
for i = n-1 to 0
  do if m(i) = 1
     then m(i) = 0
          m(i) = 1
          m(i) = 0
     else return fail
for i = n-1 to 0
  do if m(i) = 0
     then m(i) = 1
          m(i) = 0
     else return fail
return pass
```

C CODE

```
//m is the SRAM memory array with the dimension n
#define DIMENSION n
unsigned char m[DIMENSION];
bool march_b(void)
{
```

```
int i;
for(i = 0; i < DIMENSION; i++)
  m[i] = 0;
for(i = 0; i < DIMENSION; i++){
  if(m[i] == 0)
    m[i] = 1;
  else
    return false;
  if(m[i] == 1)
    m[i] = 0;
  else
    return false;
  if(m[i] == 0)
    m[i] = 1;
  else
    return false;
}
for(i = 0; i < DIMENSION; i++){
  if(m[i] == 1){
    m[i] = 0;
    m[i] = 1;
  }
  else
    return false;
}
for(i = DIMENSION-1; i >= 0; i--){
  if(m[i] == 1){
    m[i] = 0;
    m[i] = 1;
    m[i] = 0;
  }
  else
    return false;
}
for(i = DIMENSION-1; i >= 0; i--){
  if(m[i] == 0){
    m[i] = 1;
    m[i] = 0;
  }
  else
```

```
      return false;
  }
  return true;
}
```

-MARCH C (Marinescu, 1982)

```
|(w0);^(r0,w1);^(r1,w0);
|(r0);v(r0,w1);v(r1,w0);|(r0);
```

PSEUDO CODE

```
for i = 0 to n-1
  do m(i) = 0
for i = 0 to n-1
  do if m(i) = 0
     then m(i) = 1
     else return fail
for i = 0 to n-1
  do if m(i) = 1
     then m(i) = 0
     else return fail
for i = 0 to n-1
  do if m(i) = 0
     then
     else return fail
for i = n-1 to 0
  do if m(i) = 0
     then m(i) = 1
     else return fail
for i = n-1 to 0
  do if m(i) = 1
     then m(i) = 0
     else return fail
for i = 0 to n-1
  do if m(i) = 0
     then
     else return fail
return pass
```

C CODE

```
//m is the SRAM memory array with the dimension n
#define DIMENSION n
unsigned char m[DIMENSION];
bool march_c(void)
{
  int i;
  for(i = 0; i < DIMENSION; i++)
```

```
    m[i] = 0;
  for(i = 0; i < DIMENSION; i++){
    if(m[i] == 0)
      m[i] = 1;
    else
      return false;
  }
  for(i = 0; i < DIMENSION; i++){
    if(m[i] == 1)
      m[i] = 0;
    else
      return false;
  }
  for(i = 0; i < DIMENSION; i++){
    if(m[i] == 0)
      ;
    else
      return false;
  }
  for(i = DIMENSION-1; i >= 0; i--){
    if(m[i] == 0)
      m[i] = 1;
    else
      return false;
  }
  for(i = DIMENSION-1; i >= 0; i--){
    if(m[i] == 1)
      m[i] = 0;
    else
      return false;
  }
  for(i = 0; i < DIMENSION; i++){
    if(m[i] == 0)
      ;
    else
      return false;
  }
  return true;
}
```

-MOVI (de Jonge, 1976)

```
|(w0);^(r0,w1,r1);^(r1,w0,r0);v(r0,w1,r1);v(r1,w0,r0)
```

PSEUDO CODE

```
for i = 0 to n-1
  do m(i) = 0
for i = 0 to n-1
  do if m(i) = 0
     then m(i) = 1
     else return fail
     if m(i) = 1
     then
     else return fail
for i = 0 to n-1
  do if m(i) = 1
     then m(i) = 0
     else return fail
     if m(i) = 0
     then
     else return fail
for i = n-1 to 0
  do if m(i) = 0
     then m(i) = 1
     else return fail
     if m(i) = 1
     then
     else return fail
for i = n-1 to 0
  do if m(i) = 1
     then m(i) = 0
     else return fail
     if m(i) = 0
     then
     else return fail
return pass
```

C CODE

```
//m is the SRAM memory array with the dimension n
#define DIMENSION n
unsigned char m[DIMENSION];
bool movi(void)
```

```
{
  int i;
  for(i = 0; i < DIMENSION; i++)
    m[i] = 0;
  for(i = 0; i < DIMENSION; i++){
    if(m[i] == 0)
      m[i] = 1;
    else
      return false;
    if(m[i] == 1)
      ;
    else
      return false;
  }
  for(i = 0; i < DIMENSION; i++){
    if(m[i] == 1)
      m[i] = 0;
    else
      return false;
    if(m[i] == 0)
      ;
    else
      return false;
  }
  for(i = DIMENSION-1; i >= 0; i--){
    if(m[i] == 0)
      m[i] = 1;
    else
      return false;
    if(m[i] == 1)
      ;
    else
      return false;
  }
  for(i = DIMENSION-1; i >= 0; i--){
    if(m[i] == 1)
      m[i] = 0;
    else
      return false;
    if(m[i] == 0)
```

```
        ;
      else
        return false;
    }
    return true;
  }
```

-MARCH U (Van de Goor, 1997)

```
|(w0);^(r0,w1,r1,w0);^(r0,w1);v(r1,w0,r0,w1);v(r1,w0)
```

PSEUDO CODE

```
for i = 0 to n-1
  do m(i) = 0
for i = 0 to n-1
  do if m(i) = 0
     then m(i) = 1
     else return fail
     if m(i) = 1
     then m(i) = 0
     else return fail
for i = 0 to n-1
  do if m(i) = 0
     then m(i) = 1
     else return fail
for i = n-1 to 0
  do if m(i) = 1
     then m(i) = 0
     else return fail
     if m(i) = 0
     then m(i) = 1
     else return fail
for i = n-1 to 0
  do if m(i) = 1
     then m(i) = 0
     else return fail
return pass
```

C CODE

```
//m is the SRAM memory array with the dimension n
#define DIMENSION n
unsigned char m[DIMENSION];
bool march_u(void)
{
  int i;
  for(i = 0; i < DIMENSION; i++)
    m[i] = 0;
  for(i = 0; i < DIMENSION; i++){
    if(m[i] == 0)
```

```
      m[i] = 1;
    else
      return false;
    if(m[i] == 1)
      m[i] = 0;
    else
      return false;
  }
  for(i = 0; i < DIMENSION; i++){
    if(m[i] == 0)
      m[i] = 1;
    else
      return false;
  }
  for(i = DIMENSION-1; i >= 0; i--){
    if(m[i] == 1)
      m[i] = 0;
    else
      return false;
    if(m[i] == 0)
      m[i] = 1;
    else
      return false;
  }
  for(i = DIMENSION-1; i >= 0; i--){
    if(m[i] == 1)
      m[i] = 0;
    else
      return false;
  }
  return true;
}
```

-MARCH LP (Van de Goor, 1996)

```
|(w0);v(r0,w1);^(r1,w0,r0,w1);^(r1,w0);
^(r0,w1,r1,w0);^(r0)
```

PSEUDO CODE

```
for i = 0 to n-1
  do m(i) = 0
for i = n-1 to 0
  do if m(i) = 0
     then m(i) = 1
     else return fail
for i = 0 to n-1
  do if m(i) = 1
     then m(i) = 0
     else return fail
     if m(i) = 0
     then m(i) = 1
     else return fail
for i = 0 to n-1
  do if m(i) = 1
     then m(i) = 0
     else return fail
for i = 0 to n-1
  do if m(i) = 0
     then m(i) = 1
     else return fail
     if m(i) = 1
     then m(i) = 0
     else return fail
for i = 0 to n-1
  do if m(i) = 0
     then
     else return fail
return pass
```

C CODE

```
//m is the SRAM memory array with the dimension n
#define DIMENSION n
unsigned char m[DIMENSION];
bool march_u(void)
{
```

```
int i;
for(i = 0; i < DIMENSION; i++)
  m[i] = 0;
for(i = DIMENSION-1; i >= 0; i--){
  if(m[i] == 0)
    m[i] = 1;
  else
    return false;
}
for(i = 0; i < DIMENSION; i++){
  if(m[i] == 1)
    m[i] = 0;
  else
    return false;
  if(m[i] == 0)
    m[i] = 1;
  else
    return false;
}
for(i = 0; i < DIMENSION; i++){
  if(m[i] == 1)
    m[i] = 0;
  else
    return false;
}
for(i = 0; i < DIMENSION; i++){
  if(m[i] == 0)
    m[i] = 1;
  else
    return false;
  if(m[i] == 1)
    m[i] = 0;
  else
    return false;
}
for(i = 0; i < DIMENSION; i++){
  if(m[i] == 0)
    ;
  else
    return false;
    }
    return true;
}
```

-MARCH LA (Van de Goor, 1997)

```
|(w0);^(r0,w1,w0,w1,r1);^(r1,w0,w1,w0,r0);
v(r0,w1,w0,w1,r1);v(r1,w0,w1,w0,r0);v(r0)
```

PSEUDO CODE

```
for i = 0 to n-1
  do m(i) = 0
for i = 0 to n-1
  do if m(i) = 0
     then m(i) = 1
          m(i) = 0
          m(i) = 1
     else return fail
     if m(i) = 1
     then
     else return fail
for i = 0 to n-1
  do if m(i) = 1
     then m(i) = 0
          m(i) = 1
          m(i) = 0
     else return fail
     if m(i) = 0
     then
     else return fail
for i = n-1 to 0
  do if m(i) = 0
     then m(i) = 1
          m(i) = 0
          m(i) = 1
     else return fail
     if m(i) = 1
     then
     else return fail
for i = n-1 to 0
  do if m(i) = 1
     then m(i) = 0
          m(i) = 1
          m(i) = 0
     else return fail
     if m(i) = 0
```

```
        then
        else return fail
  for i = n-1 to 0
    do if m(i) = 0
        then
        else return fail
  return pass
```

C CODE

```
  //m is the SRAM memory array with the dimension n
  #define DIMENSION n
  unsigned char m[DIMENSION];
  bool march_u(void)
  {
    int i;
    for(i = 0; i < DIMENSION; i++)
      m[i] = 0;
    for(i = 0; i < DIMENSION; i++){
      if(m[i] == 0)
        m[i] = 1;
        m[i] = 0;
        m[i] = 1;
      else
        return false;
      if(m[i] == 1)
        ;
      else
        return false;
    }
    for(i = 0; i < DIMENSION; i++){
      if(m[i] == 1)
        m[i] = 0;
        m[i] = 1;
        m[i] = 0;
      else
        return false;
      if(m[i] == 0)
        ;
      else
        return false;
    }
    for(i = DIMENSION-1; i >= 0; i--){
      if(m[i] == 0)
        m[i] = 1;
        m[i] = 0;
        m[i] = 1;
      else
        return false;
      if(m[i] == 1)
        ;
      else
        return false;
    }
    for(i = DIMENSION-1; i >= 0; i--){
      if(m[i] == 1)
        m[i] = 0;
        m[i] = 1;
        m[i] = 0;
      else
        return false;
      if(m[i] == 0)
        ;
      else
        return false;
    }
    for(i = DIMENSION-1; i >= 0; i--){
      if(m[i] == 0)
        ;
      else
        return false;
    }
    return true;
  }
```

-MARCHING 1/0 Test (Breuer & Friedman, 1976)

```
^(w0);^(r0,w1,r1);v(r1,w0,r0);
^(w1);^(r1,w0,r0);v(r0,w1,r1);
```

PSEUDO CODE

```
for i = 0 to n-1
  do m(i) = 0
for i = 0 to n-1
  do if m(i) = 0
     then m(i) = 1
     else return fail
     if m(i) = 1
     then
     else return fail
for i = n-1 to 0
  do if m(i) = 1
     then m(i) = 0
     else return fail
     if m(i) = 0
     then
     else return fail
for i = 0 to n-1
  do m(i) = 1
for i = 0 to n-1
  do if m(i) = 1
     then m(i) = 0
     else return fail
     if m(i) = 0
     then
     else return fail
for i = n-1 to 0
  do if m(i) = 0
     then m(i) = 1
     else return fail
     if m(i) = 1
     then
     else return fail
return pass
```

C CODE

```
//m is the SRAM memory array with the dimension n
```

```
#define DIMENSION n
unsigned char m[DIMENSION];
bool march_u(void)
{
  int i;
  for(i = 0; i < DIMENSION; i++)
    m[i] = 0;
  for(i = 0; i < DIMENSION; i++){
    if(m[i] == 0)
      m[i] = 1;
    else
      return false;
    if(m[i] == 1)
      ;
    else
      return false;
  }
  for(i = DIMENSION-1; i >= 0; i--){
    if(m[i] == 1)
      m[i] = 0;
    else
      return false;
    if(m[i] == 0)
      ;
    else
      return false;
  }
  for(i = 0; i < DIMENSION; i++)
    m[i] = 1;
  for(i = 0; i < DIMENSION; i++){
    if(m[i] == 1)
      m[i] = 0;
    else
      return false;
    if(m[i] == 0)
      ;
    else
      return false;
  }
  for(i = DIMENSION-1; i >= 0; i--){
    if(m[i] == 0)
      m[i] = 1;
    else
      return false;
    if(m[i] == 1)
      ;
    else
      return false;
  }
  return true;
}
```

-MATS Test (Nair, Thatte & Abraham, 1979)

- MATS-ORtype

`|(w0);|(r0,w1);|(r1);`

PSEUDO CODE

```
for i = 0 to n-1
  do m(i) = 0
for i = 0 to n-1
  do if m(i) = 0
     then m(i) = 1
     else return fail
for i = 0 to n-1
  do if m(i) = 1
     then
     else return fail
return pass
```

C CODE

```
//m is the SRAM memory array with the dimension n
#define DIMENSION n
unsigned char m[DIMENSION];
bool march_u(void)
{
  int i;
  for(i = 0; i < DIMENSION; i++)
    m[i] = 0;
  for(i = 0; i < DIMENSION; i++){
    if(m[i] == 0)
      m[i] = 1;
    else
      return false;
  }
  for(i = 0; i < DIMENSION; i++){
    if(m[i] == 1)
      ;
    else
      return false;
  }
  return true;
}
```

- MATS-ANDtype

```
|(w1);|(r1,w0);|(r0);
```

PSEUDO CODE

```
for i = 0 to n-1
  do m(i) = 1
for i = 0 to n-1
  do if m(i) = 1
     then m(i) = 0
     else return fail
for i = 0 to n-1
  do if m(i) = 0
     then
     else return fail
return pass
```

C CODE

```
//m is the SRAM memory array with the dimension n
#define DIMENSION n
unsigned char m[DIMENSION];
bool march_u(void)
{
  int i;
  for(i = 0; i < DIMENSION; i++)
    m[i] = 1;
  for(i = 0; i < DIMENSION; i++){
    if(m[i] == 1)
      m[i] = 0;
    else
      return false;
  }
  for(i = 0; i < DIMENSION; i++){
    if(m[i] == 0)
      ;
    else
      return false;
  }
  return true;
}
```

-MARCH X (unpublished)

```
|(w0);^(r0,w1);v(r1,w0);|(r0);
```

PSEUDO CODE

```
for i = 0 to n-1
  do m(i) = 0
for i = 0 to n-1
  do if m(i) = 0
     then m(i) = 1
     else return fail
for i = n-1 to 0
  do if m(i) = 1
     then m(i) = 0
     else return fail
for i = 0 to n-1
  do if m(i) = 0
     then
     else return fail
return pass
```

C CODE

```
//m is the SRAM memory array with the dimension n
#define DIMENSION n
unsigned char m[DIMENSION];
bool march_u(void)
{
  int i;
  for(i = 0; i < DIMENSION; i++)
    m[i] = 0;
  for(i = 0; i < DIMENSION; i++){
    if(m[i] == 0)
      m[i] = 1;
    else
      return false;
  }
  for(i = DIMENSION - 1; i >= 0; i--){
    if(m[i] == 1)
      m[i] = 0;
    else
      return false;
  }
  for(i = 0; i < DIMENSION; i++){
    if(m[i] == 0)
      ;
    else
      return false;
  }
  return true;
}
```

-MARCH Y (unpublished)

```
|(w0);^(r0,w1,r1);v(r1,w0,r0);|(r0);
```

PSEUDO CODE

```
for i = 0 to n-1
  do m(i) = 0
for i = 0 to n-1
  do if m(i) = 0
     then m(i) = 1
     else return fail
     if m(i) = 1
     then
     else return fail
for i = n-1 to 0
  do if m(i) = 1
     then m(i) = 0
     else return fail
     if m(i) = 0
     then
     else return fail
for i = 0 to n-1
  do if m(i) = 0
     then
     else return fail
return pass
```

C CODE

```
//m is the SRAM memory array with the dimension n
#define DIMENSION n
unsigned char m[DIMENSION];
bool march_u(void)
{
  int i;
  for(i = 0; i < DIMENSION; i++)
    m[i] = 0;
  for(i = 0; i < DIMENSION; i++){
    if(m[i] == 0)
      m[i] = 1;
    else
      return false;
    if(m[i] == 1)
```

```
      ;
    else
      return false;
  }
  for(i = DIMENSION - 1; i >= 0; i--){
    if(m[i] == 1)
      m[i] = 0;
    else
      return false;
  if(m[i] == 0)
      ;
    else
      return false;
  }
  for(i = 0; i < DIMENSION; i++){
    if(m[i] == 0)
      ;
    else
      return false;
  }
  return true;
}
```

-MARCH A (Suk & Reddy, 1981)

```
|(w0);^(r0,w1,w0,w1);^(r1,w0,w1);
v(r1,w0,w1,w0);v(r0,w1,w0);
```

PSEUDO CODE

```
for i = 0 to n-1
  do m(i) = 0
for i = 0 to n-1
  do if m(i) = 0
     then m(i) = 1
          m(i) = 0
          m(i) = 1
     else return fail
for i = 0 to n-1
  do if m(i) = 1
     then m(i) = 0
          m(i) = 1
     else return fail
for i = n-1 to 0
  do if m(i) = 1
     then m(i) = 0
          m(i) = 1
          m(i) = 0
     else return fail
for i = n-1 to 0
  do if m(i) = 0
     then m(i) = 1
          m(i) = 0
     else return fail
return pass
```

C CODE

```
//m is the SRAM memory array with the dimension n
#define DIMENSION n
unsigned char m[DIMENSION];
bool march_u(void)
{
  int i;
  for(i = 0; i < DIMENSION; i++)
    m[i] = 0;
  for(i = 0; i < DIMENSION; i++){
    if(m[i] == 0)
      m[i] = 1;
      m[i] = 0;
      m[i] = 1;
    else
      return false;
  }
  for(i = 0; i < DIMENSION; i++){
    if(m[i] == 1)
      m[i] = 0;
      m[i] = 1;
    else
      return false;
  }
  for(i = DIMENSION - 1; i >= 0; i--){
    if(m[i] == 1)
      m[i] = 0;
      m[i] = 1;
      m[i] = 0;
    else
      return false;
  }
  for(i = DIMENSION - 1; i >= 0; i--){
    if(m[i] == 0)
      m[i] = 1;
      m[i] = 0;
    else
      return false;
  }
  return true;
}
```

1.2.1.17 MBIST test tables

TABLE 1.12 Fault types

Abreviation	Description
AF	address decoder fault
SAF	stuck at fault
TF	transition fault
CF	cell fault

TABLE 1.13

Abreviation	Description
L	Locate
LS	Locate some
D	Detect
DS	Detect some

-NTAb78: (Nair, Thatte, Abraham 1978)

```
    |(w0);v(r0,w1,w0,w1,r1);^(r1,w0);^(r0);^(r0,w1);
    v(r1);^(r1,w0);v(r0);v(r0,w1,w0);^(r0);^(r0);^(r0;w1,w0);
    v(r0);|(w1);v(r1,w0,w1);^(r1);^(r1,w0,w1);v(r1);
```

-MARCH G (Van de Goor, 1997)
{MARCH B};Del;|(r0,w1,r1);Del;|(r1,w0,r0);

TABLE 1.14 MBIST test

Test name	Fault types					
	AF	SAF	TF	CF	Others	Order
???	-	L	-	-		$O(n)$
Checkerboard	-	L	-	-	Refresh	$O(n)$
???	L	L	L	L	Sense amplif. rec.	$O(n^2)$

TABLE 1.14 MBIST test

???	L	L	L	L	Write recovery	$O(n^2)$
???	LS	L	L	L	Write recovery	$O(n \cdot \sqrt{n})$
GALCOL	LS	L	L	L	Write recovery	$O(n \cdot \sqrt{n})$
Sliding Diag.	LS	L	L	-		$O(n \cdot \sqrt{n})$
Butterfly						$O(n \cdot \log2(n)))$
MATS	DS	D				$O(n)$
MATS+	D	D	-	-		$O(n)$
Marching 1/0	D	D	D	-		$O(n)$
MATS++	D	D	D	-		$O(n)$
March X	D	D	D	D	Unlinked CFins	$O(n)$
March C-	D	D	D	D	Unlinked CFins	$O(n)$
March A	D	D	D	D	Unlinked CFs	$O(n)$
March Y	D	D	D	D	Linked TFs	$O(n)$
March D	D	D	D	D	Linked CFs	$O(n)$
MOVI	D	D	D	D	Read acces time	$O(n \cdot \log2(n)))$

TABLE 1.14 MBIST test

	D	D	D	D	3-coupling faults	$O(n \cdot \log 2(n)))$
	D	D	D	D	Operational faults	$O(n^2)$

1.2.1.18 MBIST test tables

TABLE 1.15

Algorithm	Address generation	Neighbor-hood cells	Fault coverage				
			SAF	TF	NPSFs		
					A	P	S
TLAPNPSF1G	G	1	L	L	L	L	L
TLAPNPSF1T	T	1	L	L	L	L	L
TLAPNPSF2T	T	2	L	L	L	L	L
TLANPSF1G	G	1	L	-	L	-	-
TLANPSF1T	T	1	L	-	L	-	-
TLANPSF2T	T	2	L	-	L	-	-
TDANPSF1G	G	1	L	-	D	-	-
TLPNPSF1G	G	1	L	L		L	-
TLPNPSF1T	T	1	L	L	-	L	-
TLPNPSF2T	T	2	L	L	-	L	-
TLSNPSF1G	G	1	L	-	-	-	L
TLSNPSF1T	T	1	L	-	-	-	L
TLSNPSF2T	T	2	L	-	-	-	L
TDSNPSF1G	G	1	L	-	-	-	D

TABLE 1.16

Abreviation	Description
G	Two-group
T	Tiling
1	Type-1
2	Type-2

TABLE 1.16

L	Locate
LS	Locate some
D	Detect
DS	Detect some

G = Two-group
T = Tiling
1 = Type-1
2 = Type-2
L = Locale
LS = Locate Some
D = Detect
DS = Desect Some

1.3 CSL MBIST Command Summary

```
csl_mbist mbist_name;
test_control(tcn);
addr_word_width(numeric_expression);
```

1.4 CSL MBIST Commands

csl_mbist *mbist_name;*

DESCRIPTION :

Declares a MBIST unit which can be customized and added to a SRAM memory.

EXAMPLE :

//small description of the example

CSL CODE

```
   //csl code goes here
```

VERILOG CODE

```
   //Verilog code goes here
```

test_control(*tcn*);

DESCRIPTION :

// description of the command

EXAMPLE :

//small description of the example

CSL CODE

```
   //csl code goes here
```

VERILOG CODE

```
   //Verilog code goes here
```

addr_word_width(*numeric_expression***);**

DESCRIPTION :

Specify the address word width explicitly or let the CSL compiler determine the number of bits required in the address generators.

EXAMPLE :

//small description of the example

CSL CODE

```
   //csl code goes here
```

VERILOG CODE

//Verilog code goes here

memory_name.**add_mbist**(*mbist_name*);

DESCRIPTION :

Adds *mbist_name* to the memory. *mbist_name* must be declared first.

EXAMPLE :

//small description of the example

CSL CODE

```
//csl code goes here
```

VERILOG CODE

```
//Verilog code goes here
```

`include_dfm ;`

DESCRIPTION :

Add additional redundant columns and logic for desig for manufacturing logic to swap out columns if they are bad with extras column(s).

EXAMPLE :

//small description of the example

CSL CODE

```
   //csl code goes here
```

VERILOG CODE

```
   //Verilog code goes here
```

```
sram_name.mbist(mbist_name);
```

DESCRIPTION :

Adds the *mbist_name* unit to the *sram_name* memory (which must be declared first).

EXAMPLE :

//small description of the example

CSL CODE

```
    //csl code goes here
```

VERILOG CODE

```
    //Verilog code goes here
```

1.5 CSL MBIST Examples

<START OF ADDED FROM TRANSCRIBED>
2006_02_24_mbist_misc.fm____

f a memory element in a teo dimensional memory is accessed by a read operation during simulation there should be an optional runtime check that will verify that the memory location has been initialized then a runtime warning should flag the reading of an uninitialized memory location

$Id $
instname=ART_RA1_272X72

```
words=272
bits=72
frequency=150
mux=4
drive=12
pipeline=
```

MemBist
window tests

TABLE 1.17

0	0	0	0	0
0	1	1	1	0
0	1	0	1	0
0	1	1	1	0
0	0	0	0	0

TABLE 1.18march0

1	1	1	1	1
1	1	1	1	1
1	1	0	1	1
1	1	1	1	1
1	1	1	1	1

TABLE 1.19march1

0	0	0	0	0
0	0	0	0	0
0	0	1	0	0
0	0	0	0	0
0	0	0	0	0

TABLE 1.20

1	1	1	1	1
1	0	0	0	1
1	0	1	0	1
1	0	0	0	1
1	1	1	1	1

MemBist
what is being tested

1. bad address decoder
2. stuck @ faults
3. neighbor cell coupling
4.

pattern generator = background
overlay bit generator
The overlay bit generator increments the x counter (which may wrap)
The x counter determines tracks where to insert 0/1 bit into the word generated by the pattern generator.
The march type specifies whether the overlay bit unit is activated meaning that the overlay bit is inserted into the pattern word.

FIGURE 1.17 Mem bist

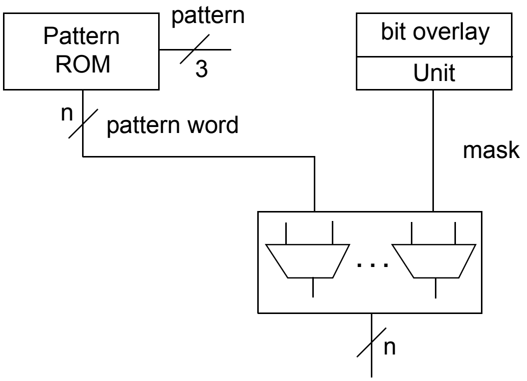


FIGURE 1.18

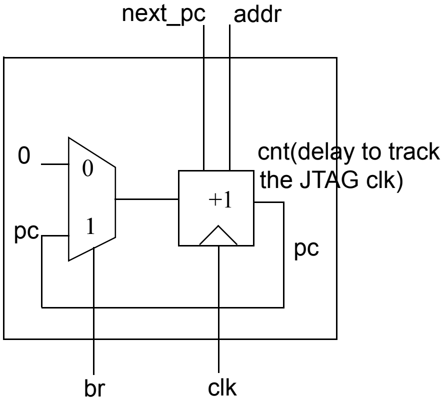


FIGURE 1.19

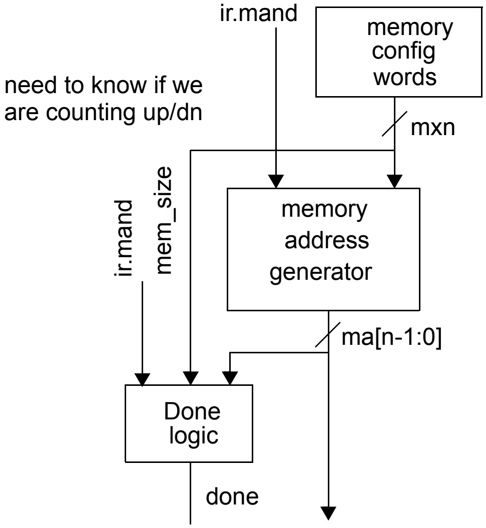
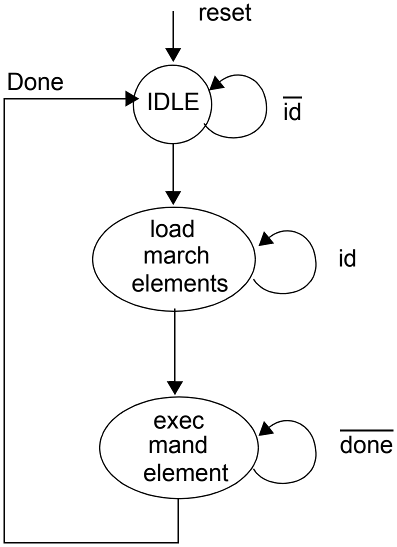


FIGURE 1.20MBist control FSM



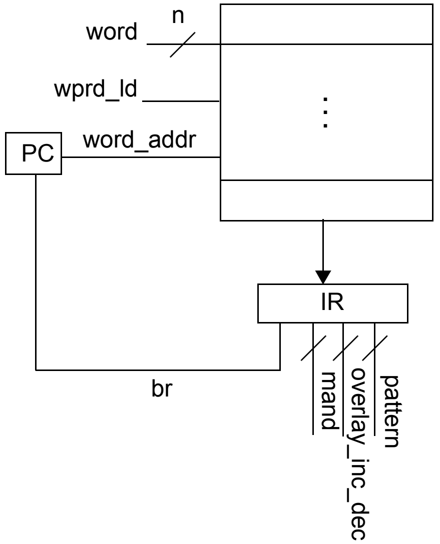


FIGURE 1.21 March Element Register File

TABLE 1.21

nwp	nrp	ww	nw	wacm	racm	psfr	psfw

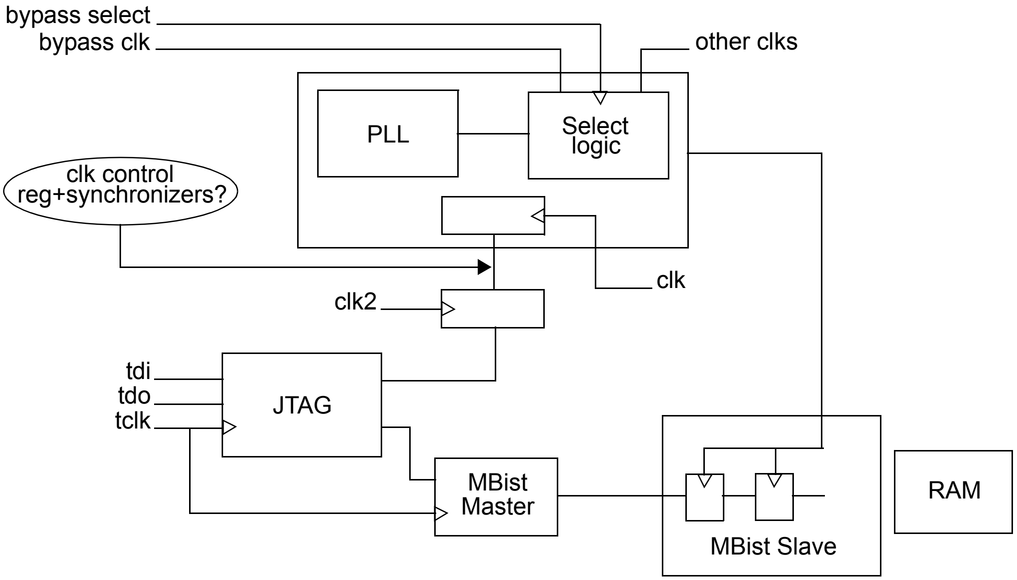
memory config register

TABLE 1.22

number write ports	number read ports	word width	number words	nwp	nrp	ww	nw

write address collision management
read address collision management
pipeline stages for read
pipeline stages for write

FIGURE 1.22


bypass select
bypass clk
other clks
PLL
Select logic
clk control reg+synchronizers?
clk
clk2
tdi
tdo
tclk
JTAG
MBist Master
MBist Slave
RAM

EDA Direct Schematic Generator
Instead of writing our own schematic generator visualization we should consider using EDA Direct Schematic Generator

Propriety

- generator
- extract
    - implicit
    - explicit
- propagate

TCL Shell
LEF/DEF
Field Solver
RC drop
.09 micron

Memory Compiler
Language
logical view - LV
physical view - PV
relationship between logical view and physical view

TABLE 1.23

test	blocks	BIST
redundancy	blocks	BISR

JTAG
SCAN CHAIN
BOUNDARY SCAN

2006_02_24_sram_bist.fm

FIGURE 1.23 Media System

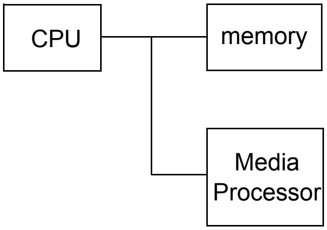
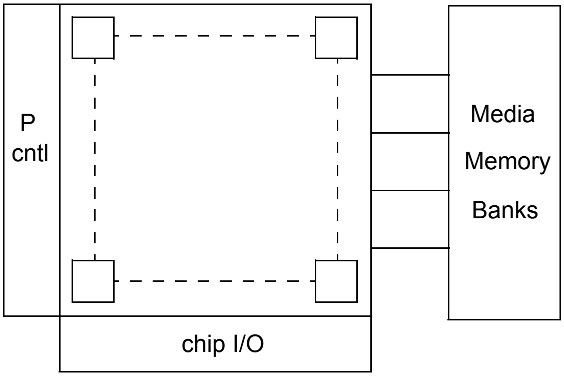


FIGURE 1.24 Media Processor



memory
BISR
BIST
ECC
IT transistors vs GT

- area difference
- speed difference

FIGURE 1.25

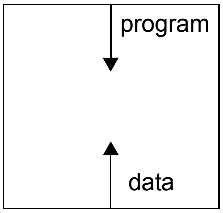
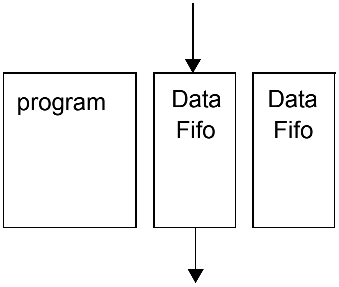


FIGURE 1.26



Spare Word Decoder

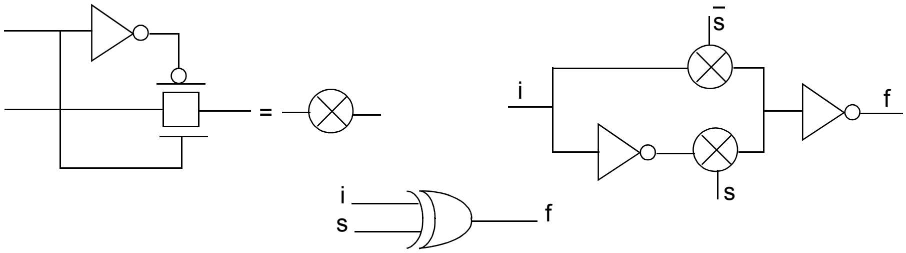
TABLE 1.24xor

a	b	f
0	0	0
0	1	1
1	0	1
1	1	0

$f= a*\bar{b}+\bar{a}*b$

this can be implemented with a pass gate

FIGURE 1.27



xor is a programmable invertor

FIGURE 1.28

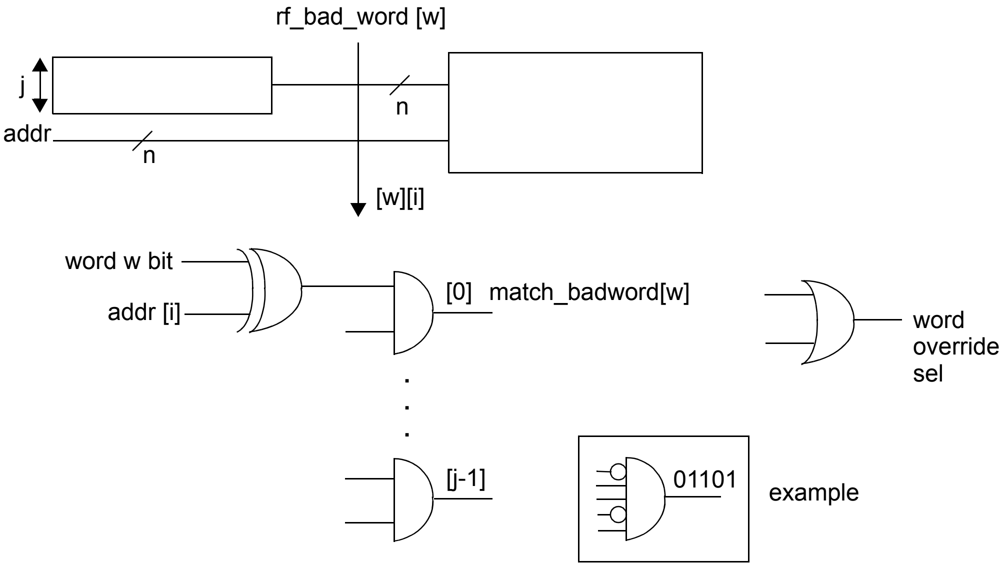


FIGURE 1.29

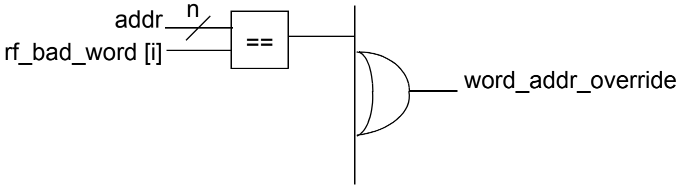
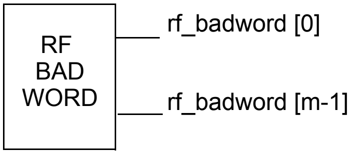


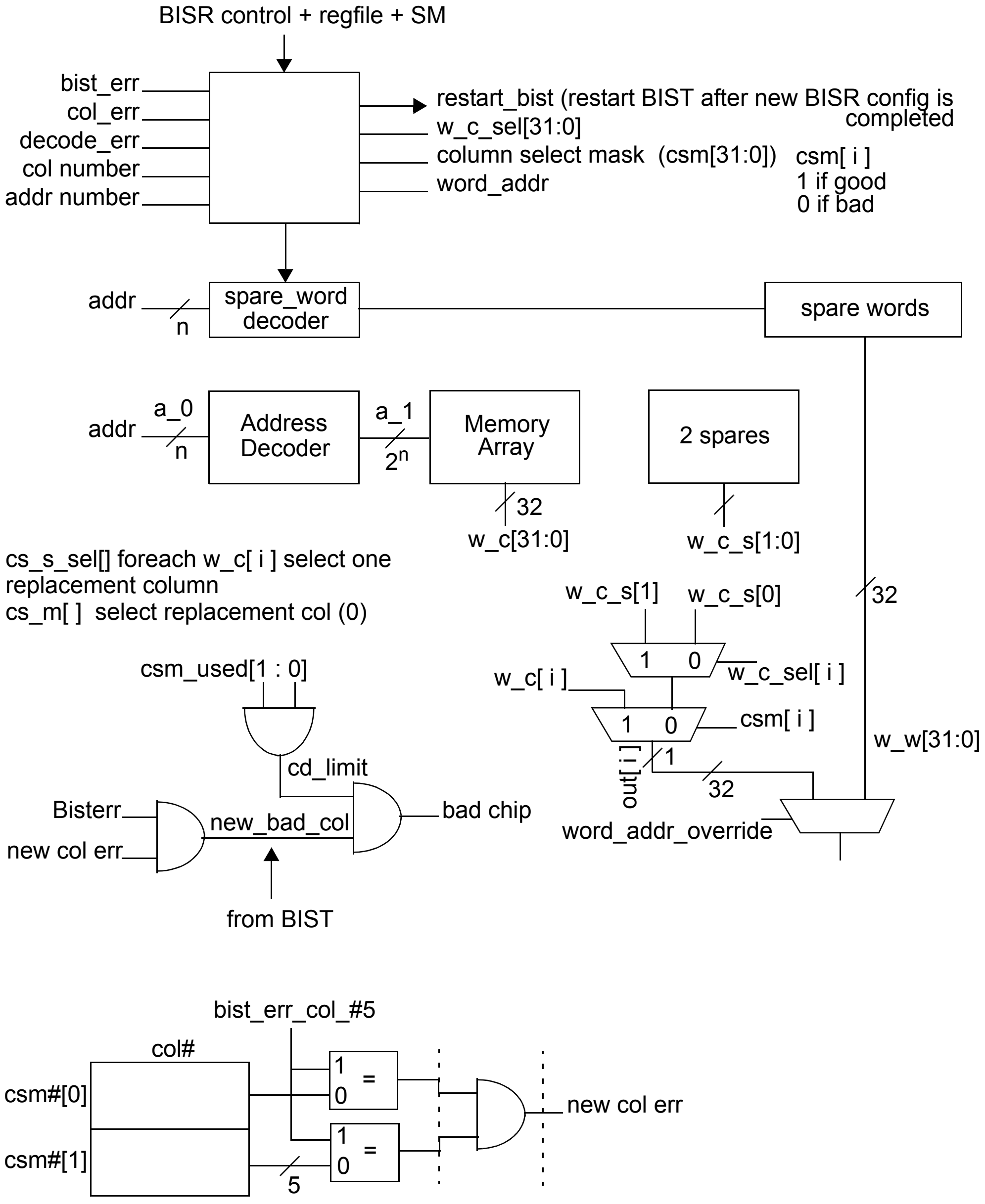
FIGURE 1.30



records the addresses in the decoder which are bad

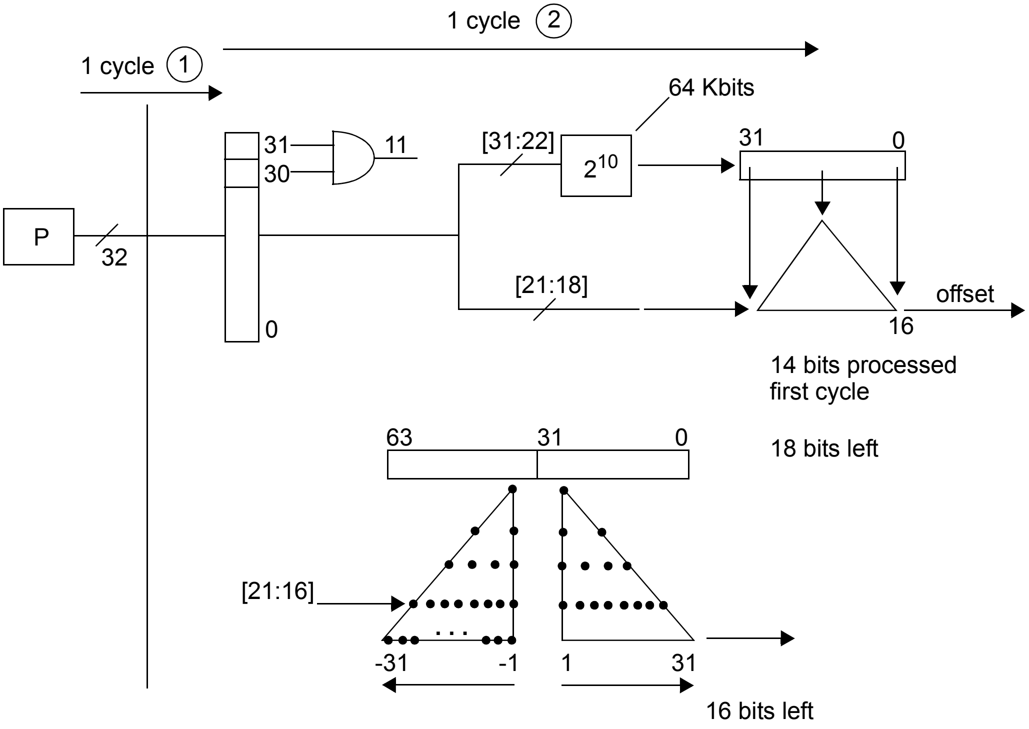
A state machine is required to handle the bist error transactions and to load the RFBAD WORD and col_sel_mask register
and w_c_s_sel register
Note that the BIST and BISR algorithms run at POR.
Note that the word_addr_override path is not a critical path the path - the path thru the array is.
The extra penalty to the access time are the output muxes.

FIGURE 1.31


BISR control + regfile + SM
bist_err
col_err
decode_err
col number
addr number
restart_bist (restart BIST after new BISR config is completed
w_c_sel[31:0]
column select mask (csm[31:0])
word_addr
csm[i]
1 if good
0 if bad
addr
n
spare_word decoder
spare words
addr
a_0
n
Address Decoder
a_1
2^n
Memory Array
32
w_c[31:0]
2 spares
w_c_s[1:0]
cs_s_sel[] foreach w_c[i] select one replacement column
cs_m[ ] select replacement col (0)
w_c_s[1]
w_c_s[0]
32
1
0
w_c_sel[i]
w_c[i]
1
0
csm[i]
w_w[31:0]
out[i]
1
32
word_addr_override
csm_used[1 : 0]
cd_limit
Bisterr
new col err
new_bad_col
bad chip
from BIST
bist_err_col_#5
col#
csm#[0]
csm#[1]
1
=
0
1
=
0
5
new col err

Back ground
Overview

FIGURE 1.32


1 cycle (2)
1 cycle (1)
64 Kbits
31
30
11
[31:22]
2^{10}
31
0
P
32
[21:18]
offset
0
16
14 bits processed first cycle
63
31
0
18 bits left
[21:16]
...
-31
-1
1
31
16 bits left

All memories tested in parallel.
Some memories may finish tests sooner than other memories.

FIGURE 1.33MBIST Master Engine

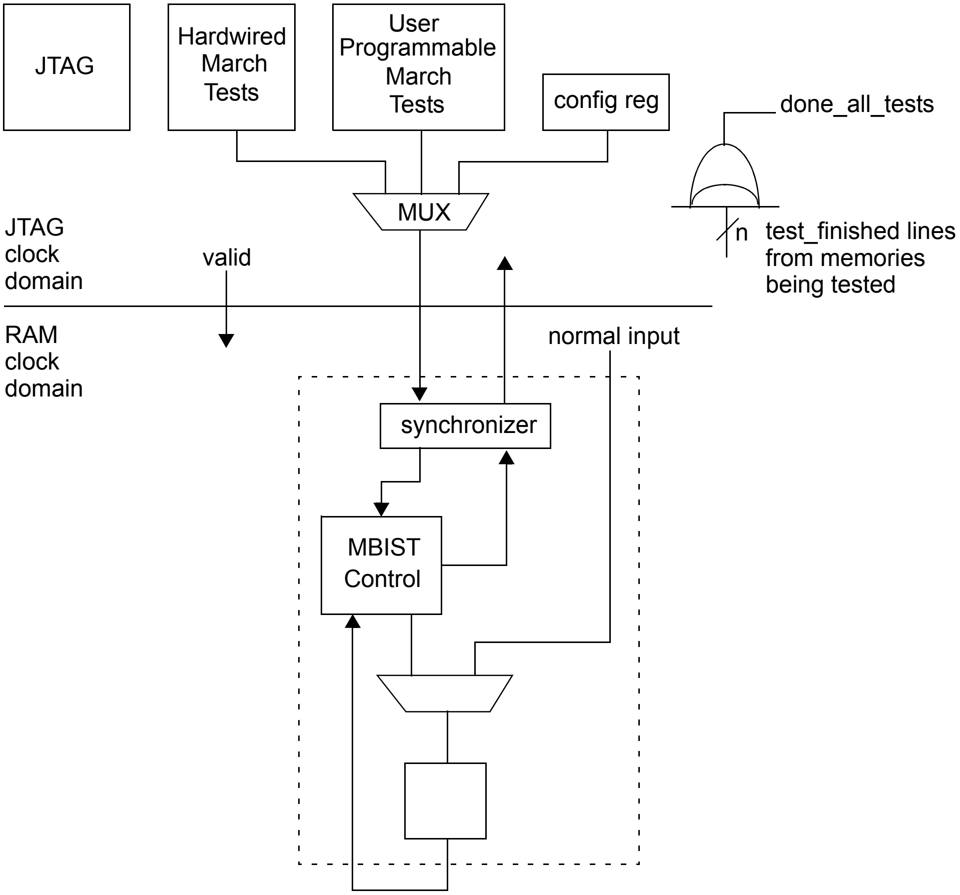
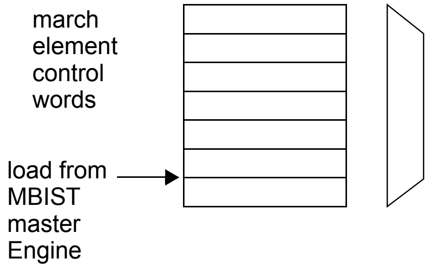


FIGURE 1.34MBIST



MBIST Data Patterns

0 ----------------- 0 1 0 ----------------- 0
1 ----------------- 1 0 1 ----------------- 1
0/1 moves through word
Back grounds

TABLE 1.25 Checker board

0	1	0	1	0	1
1	0	1	0	1	0
0	1	0	1	0	1
1	0	1	0	1	0
0	1	0	1	0	1
1	0	1	0	1	0

TABLE 1.26 Inverse checker board

1	0	1	0	1	0
0	1	0	1	0	1
1	0	1	0	1	0
0	1	0	1	0	1
1	0	1	0	1	0
0	1	0	1	0	1

0 -------------------------- 0 all 0's
1 -------------------------- 1 all 1's

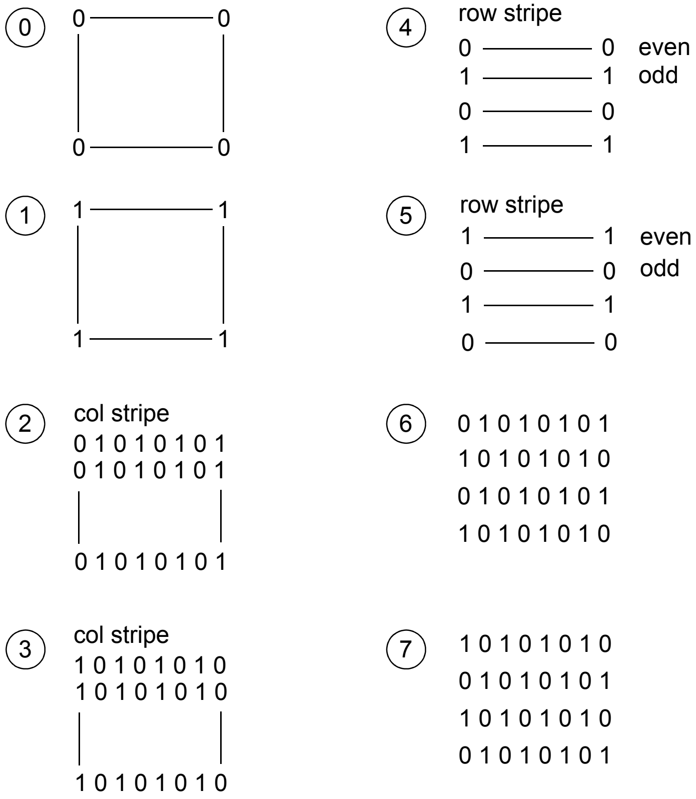
MBIST control word

Background

TABLE 1.27

RW	
0	rd
1	wr

FIGURE 1.35 Data Back ground



```
(0)  0 ---- 0
     |      |
     0 ---- 0

(1)  1 ---- 1
     |      |
     1 ---- 1

(2)  col stripe
     0 1 0 1 0 1 0 1
     0 1 0 1 0 1 0 1
     |             |
     0 1 0 1 0 1 0 1

(3)  col stripe
     1 0 1 0 1 0 1 0
     1 0 1 0 1 0 1 0
     |             |
     1 0 1 0 1 0 1 0

(4)  row stripe
     0 ---- 0  even
     1 ---- 1  odd
     0 ---- 0
     1 ---- 1

(5)  row stripe
     1 ---- 1  even
     0 ---- 0  odd
     1 ---- 1
     0 ---- 0

(6)  0 1 0 1 0 1 0 1
     1 0 1 0 1 0 1 0
     0 1 0 1 0 1 0 1
     1 0 1 0 1 0 1 0

(7)  1 0 1 0 1 0 1 0
     0 1 0 1 0 1 0 1
     1 0 1 0 1 0 1 0
     0 1 0 1 0 1 0 1
```

March LA

- single cell faults : AF, SAF, TF, SOF, DRF, RDF
- inter-word linked multicell : CFst, CFin, CFid, CFds
- inter-word linked multicell faults : TF & CFin, CFid & CFid, CFid & CFin, CFst & CFin, CFds & CFds, CFds & CFin

Full Move

- single cell faults : AF, SAF, TF, SOF, DRF, RDF
- inter-word unlinked multicell faults : CFst, CFin, CFid, CFds
- inter-word linked multicell faults : CFid & CFin
- address delay logic
- read delay logic

DBOS for rCFst

- intra-word multicell faults : rCFst, rCFin
- intra-word single cell faults : AF, SAF, TF, SOF
- 2PF 2 ar S : Do RAM allow simultaneous rd/wr access to the same address
- nPFnavs

March LR

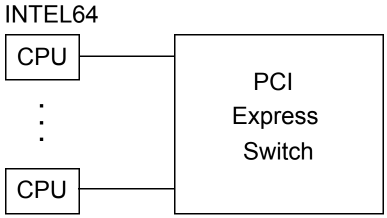
- single cell faults : AF, SAF, TF, SOF, DRF, RDF
- inter-word unlinked multicell faults : TF & CFin, CFin & CFid, CFin & CFst, CFin & CFds, CFid & CFid, CFds & CFds

Checker Board	DBStress	SAF AF TF CF
Column stripe	DBSTress	
Row stripe	DBStress	

Memory Controller for Language memory subsystems need patentable idea.
Market

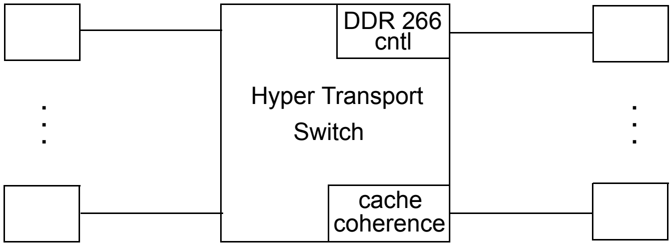
- Large memory (64 bit address) OS
- work stations
- DataBase Servers

FIGURE 1.36



is AMD already doing this?
Memory
Bus

FIGURE 1.37AMD64 transaction convertor



DIMM size ? -> #dims

Reliable
Scalable

MBIST

FIGURE 1.38 Background ROM

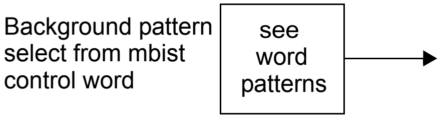


FIGURE 1.39

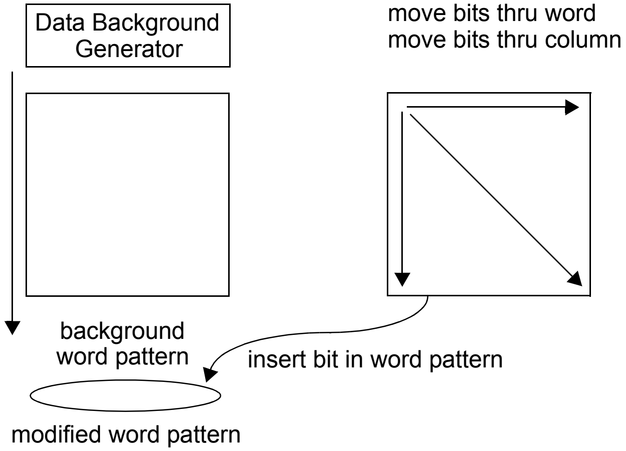
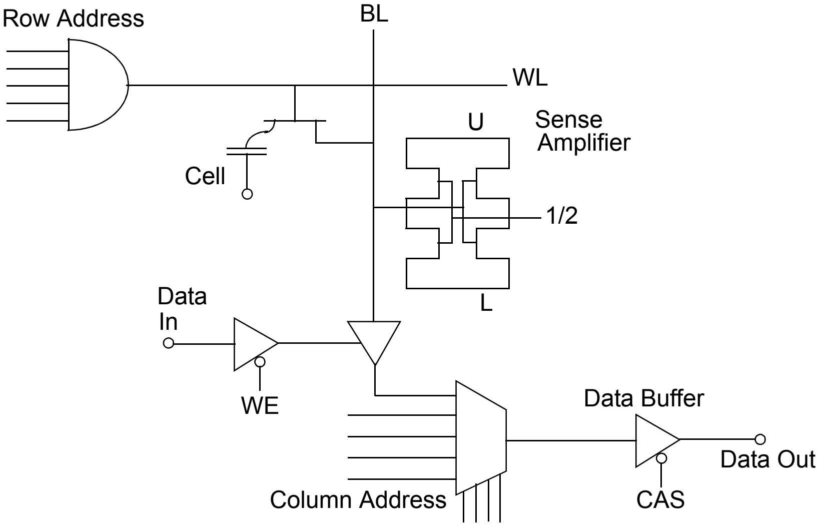

Data Background Generator
move bits thru word
move bits thru column
background word pattern
insert bit in word pattern
modified word pattern

FIGURE 1.40 Simplified DRAM Operation



Relationship between functional and reduced functional faults

TABLE 1.28 Fault table

Fault category	Fault type
SAF	Cell stuck at 0
SAF	Cell stuck at 1
SAF	Driver stuck at 0 or 1
SAF	Read line pulled down
SAF	Read line pulled up
SAF	Write line pulled down
SAF	Write line pulled up
SAF	Data line stuck
SAF	Open data line
SAF	Open data line
CF	Shorts between data lines
CF	Croostalk between data lines
AF	Address line pulled down
AF	Address line pulled up
AF	Open in address lines
AF	Shorts between address lines
AF	Open decoder
AF	Wrong access
AF	Multiple access
TF	Cell can be set to 0 But not to 1 (or vice-versa)
NPSF	Pattern sensitive iteration between cells

Coupling Fault

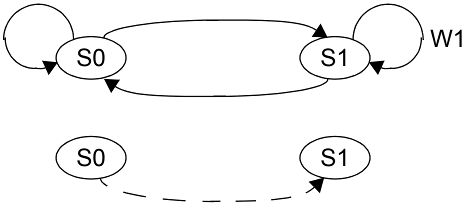
2-coupling faults

- Idempotent coupling fault
- Inversion coupling fault

K-coupling fault

Redraw the figures like in CASN!!!!

FIGURE 1.41 Neighborhood Pattern Sensitive fault



Notation of March Tests

A march element is a finite sequence of operations applied to every cell.

FIGURE 1.42

⇑ Increasing address order

⇓ Decreasing address order

A march test is a finite sequence of march elements.

FIGURE 1.43 Stuck At Fault (SAF)

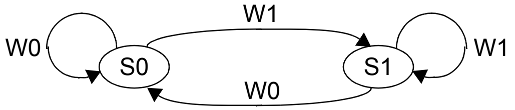


FIGURE 1.44 State diagram of a good cell

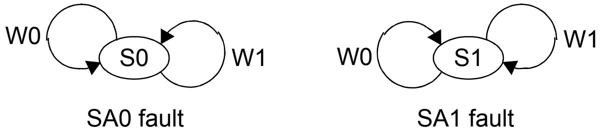


FIGURE 1.45 Transition Fault (TF)

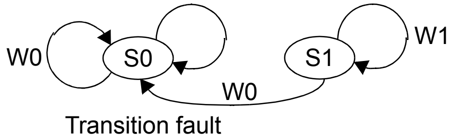


TABLE 1.29MBIST test

???	???					???	
	AF	SAF	TF	CF	Others	Order	
???	-	L	-	-		$O(n)$	
Checkerboard	-	L	-	-	Refresh	$O(n)$	
???	L	L	L	L	Sense amplif. rec.	$O(n^2)$	
???	L	L	L	L	Write recovery	$O(n^2)$	
???	LS	L	L	L	Write recovery	$O(n \cdot \sqrt{n})$	
GALCOL	LS	L	L	L	Write recovery	$O(n \cdot \sqrt{n})$	
Sliding Diag.	LS	L	L	-		$O(n \cdot \sqrt{n})$	
Butterfly						$O(n \cdot \log 2(n)))$	
MATS	DS	D				$O(n)$	
MATS+	D	D	-	-		$O(n)$	

TABLE 1.29MBIST test

Marching 1/0	D	D	D	-		$O(n)$	
MATS++	D	D	D	-		$O(n)$	
March X	D	D	D	D	Unlinked CFins	$O(n)$	
March C-	D	D	D	D	Unlinked CFins	$O(n)$	
March A	D	D	D	D	Unlinked CFs	$O(n)$	
March Y	D	D	D	D	Linked TFs	$O(n)$	
March D	D	D	D	D	Linked CFs	$O(n)$	
MOVI	D	D	D	D	Read acces time	$O(n \cdot \log 2(n)))$	
	D	D	D	D	3-coupling faults	$O(n \cdot \log 2(n)))$	
	D	D	D	D	Operational faults	$O(n^2)$	

TABLE 1.30

Algorithm	Address generation	Neighborhood cells	Fault coverage				
			SAF	TF	NPSFs		
					A	P	S
TLAPNPSF1G	G	1	L	L	L	L	L
TLAPNPSF1T	T	1	L	L	L	L	L
TLAPNPSF2T	T	2	L	L	L	L	L
TLANPSF1G	G	1	L	-	L	-	-

TABLE 1.30

TLANPSF1T	T	1	L	-	L	-	-
TLANPSF2T	T	2	L	-	L	-	-
TDANPSF1G	G	1	L	-	D	-	-
TLPNPSF1G	G	1	L	L		L	-
TLPNPSF1T	T	1	L	L	-	L	-
TLPNPSF2T	T	2	L	L	-	L	-
TLSNPSF1G	G	1	L	-	-	-	L
TLSNPSF1T	T	1	L	-	-	-	L
TLSNPSF2T	T	2	L	-	-	-	L
TDSNPSF1G	G	1	L	-	-	-	D

G = Two-group
T = Tiling
1 = Type-1
2 = Type-2
L = Locale
LS = Locate Some
D = Detect
DS = Desect Some

March Test Notation

^ = increasing address 0..n

v = decreasing address 0..n

| = don’t care, 0..n or n..0

w = w0 or w1 write data

r = r0 or r1 read data and verify

del = delay

Popular March Tests

-MATS+ (Nair, 1979)[1]

```
    |(w0);^(r0,w1);v(r1,w0);
```

-MATS++ (Van de Goor, 1991)

```
    |(w0);^(r0,w1);v(r1,w0,r0);
```

-MATS C- (Van de Goor, 1991)

```
    |(w0);^(r0,w1);^(r1,w0);v(r0,w1);v(r1,w0);|(r0)
```

-MARCH B (Suk, 1981)

```
    |(w0);^(r0,w1,r1,w0,r0,w1);^(r1,w0,w1);
    v(r1,w0,w1,w0);v(r0,w1,w0)
```

-Alg.B (Marinescu, 1982)

```
    |(w0);^(r0,w1,w0,w1);^(r1,w0,r0,w1);
    v(r1,w0,w1,w0);v(r0,w1,r1,w0)
```

-MOVI (de Jonge, 1976)

```
    |(w0);^(r0,w1,r1);^(r1,w0,r0);v(r0,w1,r1);v(r1,w0,r0)
```

-MARCH G (Van de Goor, 1997)

```
    {MARCH B};Del;|(r0,w1,r1);Del;|(r1,w0,r0);
```

-MARCH U (Van de Goor, 1997)

```
    |(w0);^(r0,w1,r1,w0);^(r0,w1);v(r1,w0,r0,w1);v(r1,w0)
```

-MARCH LP (Van de Goor, 1996)

```
    |(w0);v(r0,w1);^(r1,w0,r0,w1);^(r1,w0);
    ^(r0,w1,r1,w0);^(r0)
```

-MARCH LA (Van de Goor, 1997)

```
    |(w0);^(r0,w1,w0,w1,r1);^(r1,w0,w1,w0,r0);
    v(r0,w1,w0,w1,r1);v(r1,w0,w1,w0,r0);v(r0)
```

[1] MATS+ (Nair, 1979)

-NTAb78: (Nair, Thatte, Abraham 1978)

```
|(w0);v(r0,w1,w0,w1,r1);^(r1,w0);^(r0);^(r0,w1);
v(r1);^(r1,w0);v(r0);v(r0,w1,w0);^(r0);^(r0);^(r0;w1,w0);
v(r0);|(w1);v(r1,w0,w1);^(r1);^(r1,w0,w1);v(r1);
```

</END OF ADDED FROM TRANSCRIBED>