MX25L1605D
MX25L3205D
MX25L6405D

16M-BIT [x 1 / x 2] CMOS SERIAL FLASH
32M-BIT [x 1 / x 2] CMOS SERIAL FLASH
64M-BIT [x 1 / x 2] CMOS SERIAL FLASH

FEATURES

GENERAL

- Serial Peripheral Interface compatible -- Mode 0 and Mode 3
- 16M:16,777,216 x 1 bit structure or 8,388,608 x 2 bits (two I/O read mode) structure
  32M:33,554,432 x 1 bit structure or 16,772,216 x 2 bits (two I/O read mode) structure
  64M:67,108,864 x 1 bit structure or 33,554,432 x 2 bits (two I/O read mode) structure
- 512 Equal Sectors with 4K byte each (16Mb)
  1024 Equal Sectors with 4K byte each (32Mb)
  2048 Equal Sectors with 4K byte each (64Mb)
  - Any Sector can be erased individually
- 32 Equal Blocks with 64K byte each (16Mb)
  64 Equal Blocks with 64K byte each (32Mb)
  128 Equal Blocks with 64K byte each (64Mb)
  - Any Block can be erased individually
- Single Power Supply Operation
  - 2.7 to 3.6 volt for read, erase, and program operations
- Latch-up protected to 100mA from -1V to Vcc +1V
- Low Vcc write inhibit is from 1.5V to 2.5V

PERFORMANCE

- High Performance
  - Fast access time: 86MHz serial clock (15pF + 1TTL Load) and 66MHz serial clock (30pF + 1TTL Load)
  - Serial clock of two I/O read mode : 50MHz (15pF + TTL Load), which is equivalent to 100MHz
  - Fast program time: 1.4ms(typ.) and 5ms(max.)/page (256-byte per page)
  - Byte program time: 9us (typical)
  - Continuously program mode (automatically increase address under word program mode)
  - Fast erase time: 60ms(typ.) /sector (4K-byte per sector) ; 0.7s(typ.) /block (64K-byte per block); 14s(typ.) /chip for 16Mb, 25s(typ.) for 32Mb, and 50s(typ.) for 64Mb
- Low Power Consumption
  - Low active read current: 25mA(max.) at 86MHz, 20mA(max.) at 66MHz and 10mA(max.) at 33MHz
  - Low active programming current: 20mA (max.)
  - Low active erase current: 20mA (max.)
  - Low standby current: 20uA (max.)
  - Deep power-down mode 1uA (typical)
- Typical 100,000 erase/program cycles

SOFTWARE FEATURES

- Input Data Format
  - 1-byte Command code
- Advanced Security Features
  - Block lock protection
    The BP0-BP3 status bit defines the size of the area to be software protection against program and erase instructions
  - Additional 512-bit secured OTP for unique identifier
- Auto Erase and Auto Program Algorithm
  - Automatically erases and verifies data at selected sector
  - Automatically programs and verifies data at selected page by an internal algorithm that automatically times the program pulse widths (Any page to be programed should have page in the erased state first)

- Status Register Feature
- Electronic Identification
  - JEDEC 1-byte manufacturer ID and 2-byte device ID
  - RES command for 1-byte Device ID
  - Both REMS and REMS2 commands for 1-byte manufacturer ID and 1-byte device ID

HARDWARE FEATURES

- SCLK Input
  - Serial clock input
- SI Input
  - Serial Data Input
- SO Output
  - Serial Data Output
- WP#/ACC pin
  - Hardware write protection and program/erase acceleration
- HOLD# pin
  - pause the chip without diselecting the chip
- PACKAGE
  - 16-pin SOP (300mil)
  - 8-land WSON (8x6mm or 6x5mm)
  - 8-pin SOP (200mil, 150mil)
  - 8-pin PDIP (300mil)
  - 8-land USON (4x4mm)
  - **All Pb-free devices are RoHS Compliant**

ALTERNATIVE

- Security Serial Flash (MX25L1615D/MX25L3215D/MX25L6415D) may provides additional protection features for option. The datasheet is provided under NDA.

GENERAL DESCRIPTION

The MX25L1605D are 16,777,216 bit serial Flash memory, which is configured as 2,097,152 x 8 internally. When it is in two I/O read mode, the structure becomes 8,388,608 bits x 2. The MX25L3205D are 33,554,432 bit serial Flash memory, which is configured as 4,194,304 x 8 internally. When it is in two I/O read mode, the structure becomes 16,772,216 bits x 2. The MX25L6405D are 67,108,864 bit serial Flash memory, which is configured as 8,388,608 x 8 internally. When it is in two I/O read mode, the structure becomes 33,554,432 bits x 2. (please refer to the "Two I/O Read mode" section). The MX25L1605D/3205D/6405D feature a serial peripheral interface and software protocol allowing operation on a simple 3-wire bus. The three bus signals are a clock input (SCLK), a serial data input (SI), and a serial data output (SO). Serial access to the device is enabled by CS# input.

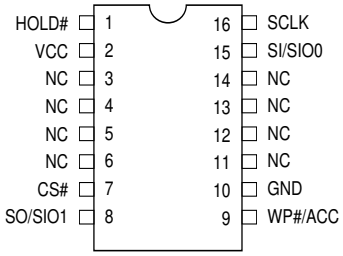
When it is in two I/O read mode, the SI pin and SO pin become SIO0 pin and SIO1 pin for address/dummy bits input and data output.

The MX25L1605D/3205D/6405D provides sequential read operation on whole chip.

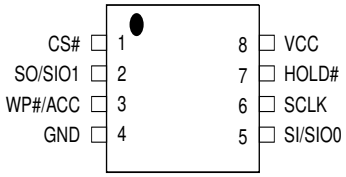
After program/erase command is issued, auto program/ erase algorithms which program/ erase and verify the specified page or sector/block locations will be executed. Program command is executed on byte basis, or page (256 bytes) basis, or word basis for Continuously program mode, and erase command is executes on sector (4K-byte), or block (64K-byte), or whole chip basis.

PIN CONFIGURATIONS

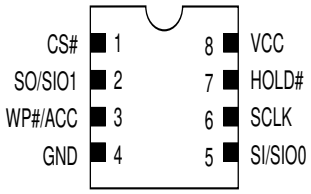
16-PIN SOP (300mil)



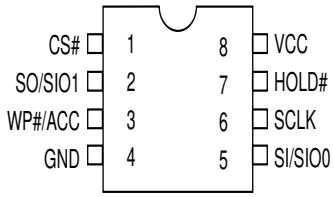
8-PIN SOP (200mil, 150mil)



8-LAND WSON (8x6mm, 6x5mm), USON (4x4mm)



8-PIN PDIP (300mil)



PACKAGE OPTIONS

	16M	32M	64M
150mil 8-SOP	V		
200mil 8-SOP	V	V	
300mil 16-SOP	V	V	V
300mil 8-PDIP	V	V	
6x5mm WSON	V	V	
8x6mm WSON			V
4x4mm USON	V	V	

PIN DESCRIPTION

SYMBOL	DESCRIPTION
CS#	Chip Select
SI/SIO0	Serial Data Input (for 1 x I/O)/ Serial Data Input & Output (for 2xI/O read mode)
SO/SIO1	Serial Data Output (for 1 x I/O)/ Serial Data Input & Output (for 2xI/O read mode)
SCLK	Clock Input
WP#/ACC	Write protection: connect to GND ; 9.5~10.5V for program/erase acceleration: connect to 9.5~10.5V
HOLD#	Hold, to pause the device without deselecting the device
VCC	+ 3.3V Power Supply
GND	Ground

To provide user with ease of interface, a status register is included to indicate the status of the chip. The status read command can be issued to detect completion status of a program or erase operation via WIP bit.

Advanced security features enhance the protection and security functions, please see security features section for more details.

When the device is not in operation and CS# is high, it is put in standby mode and draws less than 20uA DC current.

The MX25L1605D/3205D/6405D utilizes MXIC's proprietary memory cell, which reliably stores memory contents even after typical 100,000 program and erase cycles.

Table 1. Additional Feature Comparison

Additional Features / Part Name	Protection and Security		Read Performance	Identifier			
	Flexible Block protection (BP0-BP3)	512-bit secured OTP	2 I/O Read (50MHz)	Device ID (command : AB hex)	Device ID (command : 90 hex)	Device ID (command : EF hex)	RDID (command: 9F hex)
MX25L1605D	V	V	V	14 (hex)	C2 14 (hex) (if ADD=0)	C2 14 (hex) (if ADD=0)	C2 20 15 (hex)
MX25L3205D	V	V	V	15 (hex)	C2 15 (hex) (if ADD=0)	C2 15 (hex) (if ADD=0)	C2 20 16 (hex)
MX25L6405D	V	V	V	16 (hex)	C2 16 (hex) (if ADD=0)	C2 16 (hex) (if ADD=0)	C2 20 17 (hex)

Table 2. Protected Area Sizes

Status bit				Protect Level		
BP3	BP2	BP1	BP0	16Mb	32Mb	64Mb
0	0	0	0	0(none)	0(none)	0(none)
0	0	0	1	1(1block, block 31th)	1(1block, block 63th)	1(2blocks, block 126th-127th)
0	0	1	0	2(2blocks, block 30th-31th)	2(2blocks, block 62th-63th)	2(4blocks, block 124th-127th)
0	0	1	1	3(4blocks, block 28th-31th)	3(4blocks, block 60th-63th)	3(8blocks, block 120th-127th)
0	1	0	0	4(8blocks, block 24th-31th)	4(8blocks, block 56th-63th)	4(16blocks, block 112th-127th)
0	1	0	1	5(16blocks, block 16th-31th)	5(16blocks, block 48th-63th)	5(32blocks, block 96th-127th)
0	1	1	0	6(32blocks, all)	6(32blocks, block 32th-63th)	6(64blocks,block 64th-127th)
0	1	1	1	7(32blocks, all)	7(64blocks, all)	7(128blocks, all)
1	0	0	0	8(32blocks, all)	8(64blocks, all)	8(128blocks, all)
1	0	0	1	9(32blocks, all)	9(32blocks, block 0th-31th)	9(64blocks, block 0th-63th)
1	0	1	0	10(16blocks, block 0th-15th)	10(48blocks, block 0th-47th)	10(96blocks, block 0th-95th)
1	0	1	1	11(24blocks, block 0th-23th)	11(56blocks, block 0th-55th)	11(112blocks, block 0th-111th)
1	1	0	0	12(28blocks, block 0th-27th)	12(60blocks, block 0th-59th)	12(120blocks, block 0th-119th)
1	1	0	1	13(30blocks, block 0th-29th)	13(62blocks, block 0th-61th)	13(124blocks, block 0th-123th)
1	1	1	0	14(31blocks, block 0th-30th)	14(63blocks, block 0th-62th)	14(126blocks, block 0th-125th)
1	1	1	1	15(32blocks, all)	15(64blocks, all)	15(128blocks, all)

II. Additional 512-bit secured OTP for unique identifier: to provide 512-bit one-time program area for setting device unique serial number - Which may be set by factory or system customer. Please refer to table 3. 512-bit secured OTP definition.

- Security register bit 0 indicates whether the chip is locked by factory or not.
- To program the 512-bit secured OTP by entering 512-bit secured OTP mode (with ENSO command), and going through normal program procedure, and then exiting 512-bit secured OTP mode by writing EXSO command.
- Customer may lock-down the customer lockable secured OTP by writing WRSCUR(write security register) command to set customer lock-down bit1 as "1". Please refer to table of "security register definition" for security register bit definition and table of "512-bit secured OTP definition" for address range definition.
- Note: Once lock-down whatever by factory or customer, it cannot be changed any more. While in 512-bit secured OTP mode, array access is not allowed.

Table 3. 512-bit Secured OTP Definition

Address range	Size	Standard Factory Lock	Customer Lock
xxxx00~xxxx0F	128-bit	ESN (electrical serial number)	Determined by customer
xxxx10~xxxx3F	384-bit	N/A	

Table 4. COMMAND DEFINITION

COMMAND (byte)	WREN (write enable)	WRDI (write disable)	RDID (read identification)	RDSR (read status register)	WRSR (write status register)	READ (read data)	FAST READ (fast read data)	2READ (2 x I/O read command) note1	SE (sector erase)
1st byte	06 (hex)	04 (hex)	9F (hex)	05 (hex)	01 (hex)	03 (hex)	0B (hex)	BB (hex)	20 (hex)
2nd byte						AD1	AD1	ADD(2)	AD1
3rd byte						AD2	AD2	ADD(2) & Dummy(2)	AD2
4th byte						AD3	AD3		AD3
5th byte							Dummy		
Action	sets the (WEL) write enable latch bit	resets the (WEL) write enable latch bit	outputs JEDEC ID: 1-byte manufacturer ID & 2-byte device ID	to read out the values of the status register	to write new values to the status register	n bytes read out until CS# goes high	n bytes read out until CS# goes high	n bytes read out by 2 x I/O until CS# goes high	to erase the selected sector

Note 1: The count base is 4-bit for ADD(2) and Dummy(2) because of 2 x I/O. And the MSB is on SI/SIO0 which is different from 1 x I/O condition

COMMAND (byte)	BE (block erase)	CE (chip erase)	PP (Page program)	CP (Continuously program mode)	DP (Deep power down)	RDP (Release from deep power down)	RES (read electronic ID)	REMS (read electronic manufacturer & device ID)	REMS2 (read ID for 2x I/O mode)
1st byte	D8 (hex)	60 or C7 (hex)	02 (hex)	AD (hex)	B9 (hex)	AB (hex)	AB (hex)	90 (hex)	EF (hex)
2nd byte	AD1		AD1	AD1			x	x	x
3rd byte	AD2		AD2	AD2			x	x	x
4th byte	AD3		AD3	AD3			x	ADD(note 2)	ADD(note 2)
5th byte									
Action	to erase the selected block	to erase whole chip	to program the selected page	continously program whole chip, the address is automatically increase	enters deep power down mode	release from deep power down mode	to read out 1-byte device ID	outout the manufacturer ID & device ID	output the manufacturer ID & device ID

Note 2: ADD=00H will output the manufacturer ID first and ADD=01H will output device ID first

COMMAND (byte)	ENSO (enter secured OTP)	EXSO (exit secured OTP)	RDSCUR (read security register)	WRSCUR (write security register)	ESRY (enable SO to output RY/BY#)	DSRY (disable SO to output RY/BY#)
1st byte	B1 (hex)	C1 (hex)	2B (hex)	2F (hex)	70 (hex)	80 (hex)
2nd byte						
3rd byte						
4th byte						
5th byte						
Action	to enter the 512-bit secured OTP mode	to exit the 512-bit secured OTP mode	to read value of security register	to set the lock-down bit as "1" (once lock-down, cannot be updated)	to enable SO to output RY/BY# during CP mode	to disable SO to output RY/BY# during CP mode

Note 3: It is not recommoded to adopt any other code not in the command definition table, which will potentially enter the hidden mode.

Table 5-3. Memory Organization (64Mb)

Block	Sector	Address Range	
127	2047	7FF000h	7FFFFFh
	:	:	:
	2032	7F0000h	7F0FFFh
126	2031	7EF000h	7EFFFFh
	:	:	:
	2016	7E0000h	7E0FFFh
125	2015	7DF000h	7DFFFFh
	:	:	:
	2000	7D0000h	7D0FFFh
124	1999	7CF000h	7CFFFFh
	:	:	:
	1984	7C0000h	7C0FFFh
123	1983	7BF000h	7BFFFFh
	:	:	:
	1968	7B0000h	7B0FFFh
122	1967	7AF000h	7AFFFFh
	:	:	:
	1952	7A0000h	7A0FFFh
121	1951	79F000h	79FFFFh
	:	:	:
	1936	790000h	790FFFh
120	1935	78F000h	78FFFFh
	:	:	:
	1920	780000h	780FFFh
119	1919	77F000h	77FFFFh
	:	:	:
	1904	770000h	770FFFh
118	1903	76F000h	76FFFFh
	:	:	:
	1888	760000h	760FFFh
117	1887	75F000h	75FFFFh
	:	:	:
	1872	750000h	750FFFh
116	1871	74F000h	74FFFFh
	:	:	:
	1856	740000h	740FFFh
115	1855	73F000h	73FFFFh
	:	:	:
	1840	730000h	730FFFh
114	1839	72F000h	72FFFFh
	:	:	:
	1824	720000h	720FFFh
113	1823	71F000h	71FFFFh
	:	:	:
	1808	710000h	710FFFh
112	1807	70F000h	70FFFFh
	:	:	:
	1792	700000h	700FFFh
111	1791	6FF000h	6FFFFFh
	:	:	:
	1776	6F0000h	6F0FFFh
110	1775	6EF000h	6EFFFFh
	:	:	:
	1760	6E0000h	6E0FFFh
109	1759	6DF000h	6DFFFFh
	:	:	:
	1744	6D0000h	6D0FFFh
108	1743	6CF000h	6CFFFFh
	:	:	:
	1728	6C0000h	6C0FFFh
107	1727	6BF000h	6BFFFFh
	:	:	:
	1712	6B0000h	6B0FFFh
106	1711	6AF000h	6AFFFFh
	:	:	:
	1696	6A0000h	6A0FFFh
105	1695	69F000h	69FFFFh
	:	:	:
	1680	690000h	690FFFh
104	1679	68F000h	68FFFFh
	:	:	:
	1664	680000h	680FFFh
103	1663	67F000h	67FFFFh
	:	:	:
	1648	670000h	670FFFh
102	1647	66F000h	66FFFFh
	:	:	:
	1632	660000h	660FFFh
101	1631	65F000h	65FFFFh
	:	:	:
	1616	650000h	650FFFh
100	1615	64F000h	64FFFFh
	:	:	:
	1600	640000h	640FFFh
99	1599	63F000h	63FFFFh
	:	:	:
	1584	630000h	630FFFh
98	1583	62F000h	62FFFFh
	:	:	:
	1568	620000h	620FFFh
97	1567	61F000h	61FFFFh
	:	:	:
	1552	610000h	610FFFh
96	1551	60F000h	60FFFFh
	:	:	:
	1536	600000h	600FFFh

Block	Sector	Address Range	
95	1535	5FF000h	5FFFFFh
	:	:	:
	1520	5F0000h	5F0FFFh
94	1519	5EF000h	5EFFFFh
	:	:	:
	1504	5E0000h	5E0FFFh
93	1503	5DF000h	5DFFFFh
	:	:	:
	1488	5D0000h	5D0FFFh
92	1487	5CF000h	5CFFFFh
	:	:	:
	1472	5C0000h	5C0FFFh
91	1471	5BF000h	5BFFFFh
	:	:	:
	1456	5B0000h	5B0FFFh
90	1455	5AF000h	5AFFFFh
	:	:	:
	1440	5A0000h	5A0FFFh
89	1439	59F000h	59FFFFh
	:	:	:
	1424	590000h	590FFFh
88	1423	58F000h	58FFFFh
	:	:	:
	1408	580000h	580FFFh
87	1407	57F000h	57FFFFh
	:	:	:
	1392	570000h	570FFFh
86	1391	56F000h	56FFFFh
	:	:	:
	1376	560000h	560FFFh
85	1375	55F000h	55FFFFh
	:	:	:
	1360	550000h	550FFFh
84	1359	54F000h	54FFFFh
	:	:	:
	1344	540000h	540FFFh
83	1343	53F000h	53FFFFh
	:	:	:
	1328	530000h	530FFFh
82	1327	52F000h	52FFFFh
	:	:	:
	1312	520000h	520FFFh
81	1311	51F000h	51FFFFh
	:	:	:
	1296	510000h	510FFFh
80	1295	50F000h	50FFFFh
	:	:	:
	1280	500000h	500FFFh

Block	Sector	Address Range	
79	1279	4FF000h	4FFFFFh
	:	:	:
	1264	4F0000h	4F0FFFh
78	1263	4EF000h	4EFFFFh
	:	:	:
	1248	4E0000h	4E0FFFh
77	1247	4DF000h	4DFFFFh
	:	:	:
	1232	4D0000h	4D0FFFh
76	1231	4CF000h	4CFFFFh
	:	:	:
	1216	4C0000h	4C0FFFh
75	1215	4BF000h	4BFFFFh
	:	:	:
	1200	4B0000h	4B0FFFh
74	1119	4AF000h	4AFFFFh
	:	:	:
	1184	4A0000h	4A0FFFh
73	1183	49F000h	49FFFFh
	:	:	:
	1168	490000h	490FFFh
72	1167	48F000h	48FFFFh
	:	:	:
	1152	480000h	480FFFh
71	1151	47F000h	47FFFFh
	:	:	:
	1136	470000h	470FFFh
70	1135	46F000h	46FFFFh
	:	:	:
	1120	460000h	460FFFh
69	1119	45F000h	45FFFFh
	:	:	:
	1104	450000h	450FFFh
68	1103	44F000h	44FFFFh
	:	:	:
	1088	440000h	440FFFh
67	1087	43F000h	43FFFFh
	:	:	:
	1072	430000h	430FFFh
66	1071	42F000h	42FFFFh
	:	:	:
	1056	420000h	420FFFh
65	1055	41F000h	41FFFFh
	:	:	:
	1040	410000h	410FFFh
64	1039	40F000h	40FFFFh
	:	:	:
	1024	400000h	400FFFh

Block	Sector	Address Range	
63	1023	3FF000h	3FFFFFh
	:	:	:
	1008	3F0000h	3F0FFFh
62	1007	3EF000h	3EFFFFh
	:	:	:
	992	3E0000h	3E0FFFh
61	991	3DF000h	3DFFFFh
	:	:	:
	976	3D0000h	3D0FFFh
60	975	3CF000h	3CFFFFh
	:	:	:
	960	3C0000h	3C0FFFh
59	959	3BF000h	3BFFFFh
	:	:	:
	944	3B0000h	3B0FFFh
58	943	3AF000h	3AFFFFh
	:	:	:
	928	3A0000h	3A0FFFh
57	927	39F000h	39FFFFh
	:	:	:
	912	390000h	390FFFh
56	911	38F000h	38FFFFh
	:	:	:
	896	380000h	380FFFh
55	895	37F000h	37FFFFh
	:	:	:
	880	370000h	370FFFh
54	879	36F000h	36FFFFh
	:	:	:
	864	360000h	360FFFh
53	863	35F000h	35FFFFh
	:	:	:
	848	350000h	350FFFh
52	847	34F000h	34FFFFh
	:	:	:
	832	340000h	340FFFh
51	831	33F000h	33FFFFh
	:	:	:
	816	330000h	330FFFh
50	815	32F000h	32FFFFh
	:	:	:
	800	320000h	320FFFh
49	799	31F000h	31FFFFh
	:	:	:
	784	310000h	310FFFh
48	783	30F000h	30FFFFh
	:	:	:
	768	300000h	300FFFh

Block	Sector	Address Range	
47	767	2FF000h	2FFFFFh
	:	:	:
	752	2F0000h	2F0FFFh
46	751	2EF000h	2EFFFFh
	:	:	:
	736	2E0000h	2E0FFFh
45	735	2DF000h	2DFFFFh
	:	:	:
	720	2D0000h	2D0FFFh
44	719	2CF000h	2CFFFFh
	:	:	:
	704	2C0000h	2C0FFFh
43	703	2BF000h	2BFFFFh
	:	:	:
	688	2B0000h	2B0FFFh
42	687	2AF000h	2AFFFFh
	:	:	:
	672	2A0000h	2A0FFFh
41	671	29F000h	29FFFFh
	:	:	:
	656	290000h	290FFFh
40	655	28F000h	28FFFFh
	:	:	:
	640	280000h	280FFFh
39	639	27F000h	27FFFFh
	:	:	:
	624	270000h	270FFFh
38	623	26F000h	26FFFFh
	:	:	:
	608	260000h	260FFFh
37	607	25F000h	25FFFFh
	:	:	:
	592	250000h	250FFFh
36	591	24F000h	24FFFFh
	:	:	:
	576	240000h	240FFFh
35	575	23F000h	23FFFFh
	:	:	:
	560	230000h	230FFFh
34	559	22F000h	22FFFFh
	:	:	:
	544	220000h	220FFFh
33	543	21F000h	21FFFFh
	:	:	:
	528	210000h	210FFFh
32	527	20F000h	20FFFFh
	:	:	:
	512	200000h	200FFFh

Block	Sector	Address Range	
31	511	1FF000h	1FFFFFh
	:	:	:
	496	1F0000h	1F0FFFh
30	495	1EF000h	1EFFFFh
	:	:	:
	480	1E0000h	1E0FFFh
29	479	1DF000h	1DFFFFh
	:	:	:
	464	1D0000h	1D0FFFh
28	463	1CF000h	1CFFFFh
	:	:	:
	448	1C0000h	1C0FFFh
27	447	1BF000h	1BFFFFh
	:	:	:
	432	1B0000h	1B0FFFh
26	431	1AF000h	1AFFFFh
	:	:	:
	416	1A0000h	1A0FFFh
25	415	19F000h	19FFFFh
	:	:	:
	400	190000h	190FFFh
24	399	18F000h	18FFFFh
	:	:	:
	384	180000h	180FFFh
23	383	17F000h	17FFFFh
	:	:	:
	368	170000h	170FFFh
22	367	16F000h	16FFFFh
	:	:	:
	352	160000h	160FFFh
21	351	15F000h	15FFFFh
	:	:	:
	336	150000h	150FFFh
20	335	14F000h	14FFFFh
	:	:	:
	320	140000h	140FFFh
19	319	13F000h	13FFFFh
	:	:	:
	304	130000h	130FFFh
18	303	12F000h	12FFFFh
	:	:	:
	288	120000h	120FFFh
17	287	11F000h	11FFFFh
	:	:	:
	272	110000h	110FFFh
16	271	10F000h	10FFFFh
	:	:	:
	256	100000h	100FFFh

Block	Sector	Address Range	
15	255	0FF000h	0FFFFFh
	:	:	:
	240	0F0000h	0F0FFFh
14	239	0EF000h	0EFFFFh
	:	:	:
	224	0E0000h	0E0FFFh
13	223	0DF000h	0DFFFFh
	:	:	:
	208	0D0000h	0D0FFFh
12	207	0CF000h	0CFFFFh
	:	:	:
	192	0C0000h	0C0FFFh
11	191	0BF000h	0BFFFFh
	:	:	:
	176	0B0000h	0B0FFFh
10	175	0AF000h	0AFFFFh
	:	:	:
	160	0A0000h	0A0FFFh
9	159	09F000h	09FFFFh
	:	:	:
	144	090000h	090FFFh
8	143	08F000h	08FFFFh
	:	:	:
	128	080000h	080FFFh
7	127	07F000h	07FFFFh
	:	:	:
	112	070000h	070FFFh
6	111	06F000h	06FFFFh
	:	:	:
	96	060000h	060FFFh
5	95	05F000h	05FFFFh
	:	:	:
	80	050000h	050FFFh
4	79	04F000h	04FFFFh
	:	:	:
	64	040000h	040FFFh
3	63	03F000h	03FFFFh
	:	:	:
	48	030000h	030FFFh
2	47	02F000h	02FFFFh
	:	:	:
	32	020000h	020FFFh
1	31	01F000h	01FFFFh
	:	:	:
	16	010000h	010FFFh
0	15	00F000h	00FFFFh
	:	:	:
	4	004000h	004FFFh
	3	003000h	003FFFh
	2	002000h	002FFFh
	1	001000h	001FFFh
	0	000000h	000FFFh

Note: If SRWD bit=1 but WP#/ACC is low, it is impossible to write the Status Register even if the WEL bit has previously been set. It is rejected to write the Status Register and not be executed.

Hardware Protected Mode (HPM):

- When SRWD bit=1, and then WP#/ACC is low (or WP#/ACC is low before SRWD bit=1), it enters the hardware protected mode (HPM). The data of the protected area is protected by software protected mode by BP3, BP2, BP1, BP0 and hardware protected mode by the WP#/ACC to against data modification.

Note: to exit the hardware protected mode requires WP#/ACC driving high once the hardware protected mode is entered. If the WP#/ACC pin is permanently connected to high, the hardware protected mode can never be entered; only can use software protected mode via BP3, BP2, BP1, BP0.

(6) Read Data Bytes (READ)

The read instruction is for reading data out. The address is latched on rising edge of SCLK, and data shifts out on the falling edge of SCLK at a maximum frequency fR. The first address byte can be at any location. The address is automatically increased to the next higher address after each byte data is shifted out, so the whole memory can be read out at a single READ instruction. The address counter rolls over to 0 when the highest address has been reached.

The sequence of issuing READ instruction is: CS# goes low-> sending READ instruction code-> 3-byte address on SI -> data out on SO-> to end READ operation can use CS# to high at any time during data out. (see Figure. 17)

(7) Read Data Bytes at Higher Speed (FAST_READ)

The FAST_READ instruction is for quickly reading data out. The address is latched on rising edge of SCLK, and data of each bit shifts out on the falling edge of SCLK at a maximum frequency fC. The first address byte can be at any location. The address is automatically increased to the next higher address after each byte data is shifted out, so the whole memory can be read out at a single FAST_READ instruction. The address counter rolls over to 0 when the highest address has been reached.

The sequence of issuing FAST_READ instruction is: CS# goes low-> sending FAST_READ instruction code-> 3-byte address on SI-> 1-dummy byte address on SI->data out on SO-> to end FAST_READ operation can use CS# to high at any time during data out. (see Figure. 18)

While Program/Erase/Write Status Register cycle is in progress, FAST_READ instruction is rejected without any impact on the Program/Erase/Write Status Register current cycle.

(8) 2 x I/O Read Mode (2READ)

The 2READ instruction enable double throughput of Serial Flash in read mode. The address is latched on rising edge of SCLK, and data of every two bits(interleave on 2 I/O pins) shift out on the falling edge of SCLK at a maximum frequency fT. The first address byte can be at any location. The address is automatically increased to the next higher address after each byte data is shifted out, so the whole memory can be read out at a single 2READ instruction. The address counter rolls over to 0 when the highest address has been reached. Once writing 2READ instruction, the following address/dummy/data out will perform as 2-bit instead of previous 1-bit.

The sequence of issuing 2READ instruction is: CS# goes low→sending 2READ instruction→24-bit address interleave on SIO1 & SIO0→ 8-bit dummy interleave on SIO1 & SIO0→ data out interleave on SIO1 & SIO0→ to end 2READ operation can use CS# to high at any time during data out (see Figure of 2 x I/O Read Mode Timing Waveform)

While Program/Erase/Write Status Register cycle is in progress, 2READ instruction is rejected without any impact on the Program/Erase/Write Status Register current cycle.

The 2 I/O only perform read operation. Program/Erase /Read ID/Read status/Read ID....operation do not support 2 I/O throughputs.

(9) Sector Erase (SE)

The Sector Erase (SE) instruction is for erasing the data of the chosen sector to be "1". The instruction is used for any 4K-byte sector. A Write Enable (WREN) instruction must execute to set the Write Enable Latch (WEL) bit before sending the Sector Erase (SE). Any address of the sector (see table 3) is a valid address for Sector Erase (SE) instruction. The CS# must go high exactly at the byte boundary (the latest eighth of address byte been latched-in); otherwise, the instruction will be rejected and not executed.

Address bits [Am-A12] (Am is the most significant address) select the sector address.

The sequence of issuing SE instruction is: CS# goes low -> sending SE instruction code-> 3-byte address on SI -> CS# goes high. (see Figure 22)

The self-timed Sector Erase Cycle time (tSE) is initiated as soon as Chip Select (CS#) goes high. The Write in Progress (WIP) bit still can be check out during the Sector Erase cycle is in progress. The WIP sets 1 during the tSE timing, and sets 0 when Sector Erase Cycle is completed, and the Write Enable Latch (WEL) bit is reset. If the page is protected by BP3, BP2, BP1, BP0 bits, the Sector Erase (SE) instruction will not be executed on the page.

(10) Block Erase (BE)

The Block Erase (BE) instruction is for erasing the data of the chosen block to be "1". The instruction is used for 64K-byte sector erase operation. A Write Enable (WREN) instruction must execute to set the Write Enable Latch (WEL) bit before sending the Block Erase (BE). Any address of the block (see table 3) is a valid address for Block Erase (BE) instruction. The CS# must go high exactly at the byte boundary (the latest eighth of address byte been latched-in); otherwise, the instruction will be rejected and not executed.

The sequence of issuing BE instruction is: CS# goes low -> sending BE instruction code-> 3-byte address on SI -> CS# goes high. (see Figure 23)

The self-timed Block Erase Cycle time (tBE) is initiated as soon as Chip Select (CS#) goes high. The Write in Progress (WIP) bit still can be check out during the Sector Erase cycle is in progress. The WIP sets 1 during the tBE timing, and sets 0 when Sector Erase Cycle is completed, and the Write Enable Latch (WEL) bit is reset. If the page is protected by BP3, BP2, BP1, BP0 bits, the Block Erase (BE) instruction will not be executed on the page.

(11) Chip Erase (CE)

The Chip Erase (CE) instruction is for erasing the data of the whole chip to be "1". A Write Enable (WREN) instruction must execute to set the Write Enable Latch (WEL) bit before sending the Chip Erase (CE). Any address of the sector (see table 3) is a valid address for Chip Erase (CE) instruction. The CS# must go high exactly at the byte boundary(the latest eighth of address byte been latched-in); otherwise, the instruction will be rejected and not executed.

The sequence of issuing CE instruction is: CS# goes low-> sending CE instruction code-> CS# goes high. (see Figure 24)

The self-timed Chip Erase Cycle time (tCE) is initiated as soon as Chip Select (CS#) goes high. The Write in Progress (WIP) bit still can be check out during the Chip Erase cycle is in progress. The WIP sets 1 during the tCE timing, and sets 0 when Chip Erase Cycle is completed, and the Write Enable Latch (WEL) bit is reset. If the chip is protected by BP3, BP2, BP1, BP0 bits, the Chip Erase (CE) instruction will not be executed. It will be only executed when BP3, BP2, BP1, BP0 all set to "0".

(12) Page Program (PP)

The Page Program (PP) instruction is for programming the memory to be "0". A Write Enable (WREN) instruction must execute to set the Write Enable Latch (WEL) bit before sending the Page Program (PP). If the eight least significant address bits (A7-A0) are not all 0, all transmitted data which goes beyond the end of the current page are programmed from the start address if the same page (from the address whose 8 least significant address bits (A7-A0) are all 0). The CS# must keep during the whole Page Program cycle. The CS# must go high exactly at the byte boundary(the latest eighth of address byte been latched-in); otherwise, the instruction will be rejected and not executed. If more than 256 bytes are sent to the device, the data of the last 256-byte is programmed at the request page and previous data will be disregarded. If less than 256 bytes are sent to the device, the data is programmed at the request address of the page without effect on other address of the same page.

The sequence of issuing PP instruction is: CS# goes low-> sending PP instruction code-> 3-byte address on SI-> at least 1-byte on data on SI-> CS# goes high. (see Figure 20)

The self-timed Page Program Cycle time (tPP) is initiated as soon as Chip Select (CS#) goes high. The Write in Progress (WIP) bit still can be check out during the Page Program cycle is in progress. The WIP sets 1 during the tPP timing, and sets 0 when Page Program Cycle is completed, and the Write Enable Latch (WEL) bit is reset. If the page is protected by BP3, BP2, BP1, BP0 bits, the Page Program (PP) instruction will not be executed.

(13) Continuously program mode (CP mode)

The CP mode may enhance program performance by automatically increasing address to the next higher address after each byte data has been programmed.

The Continuously program (CP) instruction is for multiple byte program to Flash. A write Enable (WREN) instruction must execute to set the Write Enable Latch(WEL) bit before sending the Continuously program (CP) instruction. CS# requires to go high before CP instruction is executing. After CP instruction and address input, two bytes of data is input sequentially from MSB(bit7) to LSB(bit0). The first byte data will be programmed to the initial address range with A0=0 and second byte data with A0=1. If only one byte data is input, the CP mode will not process. If more than two bytes data are input, the additional data will be ignored and only two byte data are valid. The CP program instruction will be ignored and not affect the WEL bit if it is applied to a protected memory area. Any byte to be programmed should be in the erase state (FF) first. It will not roll over during the CP mode, once the last unprotected address has been reached, the chip will exit CP mode and reset write Enable Latch bit (WEL) as "0" and CP mode bit as "0". Please check the WIP bit status if it is not in write progress before entering next valid instruction. During CP mode, the valid commands are CP command (AD hex), WRDI command (04 hex), RDSR command (05 hex), RDPR command (A1 hex), and RDSCUR command (2B hex). And the WRDI command is valid after completion of a CP programming cycle, which means the WIP bit=0.

The sequence of issuing CP instruction is : CS# high to low-> sending CP instruction code-> 3-byte address on SI-> Data Byte on SI->CS# goes high to low-> sending CP instruction.....-> last desired byte programmed or sending Write Disable (WRDI) instruction to end CP mode-> sending RDSR instruction to verify if CP mode is ended. (see Figure of CP mode timing waveform)

Three methods to detect the completion of a program cycle during CP mode:
1) Software method-I: by checking WIP bit of Status Register to detect the completion of CP mode.

2) Software method-II: by waiting for a tBP time out to determine if it may load next valid command or not.
3) Hardware method: by writing ESRY (enable SO to output RY/BY#) instruction to detect the completion of a program cycle during CP mode. The ESRY instruction must be executed before CP mode execution. Once it is enable in CP mode, the CS# goes low will drive out the RY/BY# status on SO, "0" indicates busy stage, "1" indicates ready stage, SO pin outputs tri-state if CS# goes high. DSRY (disable SO to output RY/BY#) instruction to disable the SO to output RY/BY# and return to status register data output during CP mode. Please note that the ESRY/DSRY command are not accepted unless the completion of CP mode.

(14) Deep Power-down (DP)

The Deep Power-down (DP) instruction is for setting the device on the minimizing the power consumption (to entering the Deep Power-down mode), the standby current is reduced from ISB1 to ISB2). The Deep Power-down mode requires the Deep Power-down (DP) instruction to enter, during the Deep Power-down mode, the device is not active and all Write/Program/Erase instruction are ignored. When CS# goes high, it's only in standby mode not deep power-down mode. It's different from Standby mode.

The sequence of issuing DP instruction is: CS# goes low-> sending DP instruction code-> CS# goes high. (see Figure 25)

Once the DP instruction is set, all instruction will be ignored except the Release from Deep Power-down mode (RDP) and Read Electronic Signature (RES) instruction. (those instructions allow the ID being reading out). When Power-down, the deep power-down mode automatically stops, and when power-up, the device automatically is in standby mode. For RDP instruction the CS# must go high exactly at the byte boundary (the latest eighth bit of instruction code been latched-in); otherwise, the instruction will not executed. As soon as Chip Select (CS#) goes high, a delay of tDP is required before entering the Deep Power-down mode and reducing the current to ISB2.

(15) Release from Deep Power-down (RDP), Read Electronic Signature (RES)

The Release from Deep Power-down (RDP) instruction is terminated by driving Chip Select (CS#) High. When Chip Select (CS#) is driven High, the device is put in the Stand-by Power mode. If the device was not previously in the Deep Power-down mode, the transition to the Stand-by Power mode is immediate. If the device was previously in the Deep Power-down mode, though, the transition to the Stand-by Power mode is delayed by tRES2, and Chip Select (CS#) must remain High for at least tRES2(max), as specified in Table 6. Once in the Stand-by Power mode, the device waits to be selected, so that it can receive, decode and execute instructions.

RES instruction is for reading out the old style of 8-bit Electronic Signature, whose values are shown as table of ID Definitions. This is not the same as RDID instruction. It is not recommended to use for new design. For new design, please use RDID instruction. Even in Deep power-down mode, the RDP and RES are also allowed to be executed, only except the device is in progress of program/erase/write cycle; there's no effect on the current program/erase/write cycle in progress.

The sequence is shown as Figure 26,27.

The RES instruction is ended by CS# goes high after the ID been read out at least once. The ID outputs repeatedly if continuously send the additional clock cycles on SCLK while CS# is at low. If the device was not previously in Deep Power-down mode, the device transition to standby mode is immediate. If the device was previously in Deep Power-down mode, there's a delay of tRES2 to transit to standby mode, and CS# must remain to high at least tRES2(max). Once in the standby mode, the device waits to be selected, so it can be receive, decode, and execute instruction.

The RDP instruction is for releasing from Deep Power Down Mode.

(16) Read Electronic Manufacturer ID & Device ID (REMS), (REMS2)

The REMS & REMS2 instruction is an alternative to the Release from Power-down/Device ID instruction that provides both the JEDEC assigned manufacturer ID and the specific device ID.

The REMS & REMS2 instruction is very similar to the Release from Power-down/Device ID instruction. The instruction is initiated by driving the CS# pin low and shift the instruction code "90h" or "EFh" followed by two dummy bytes and one bytes address (A7~A0). After which, the Manufacturer ID for MXIC (C2h) and the Device ID are shifted out on the falling edge of SCLK with most significant bit (MSB) first as shown in figure 25. The Device ID values are listed in Table of ID Definitions. If the one-byte address is initially set to 01h, then the device ID will be read first and then followed by the Manufacturer ID. The Manufacturer and Device IDs can be read continuously, alternating from one to the other. The instruction is completed by driving CS# high.

Table 7. ID Definitions

Command Type	MX25L1605D			MX25L3205D			MX25L6405D		
RDID (JEDEC ID)	Manufacturer ID	Memory type	Memory Density	Manufacturer ID	Memory type	Memory Density	Manufacturer ID	Memory type	Memory Density
	C2	20	15	C2	20	16	C2	20	17
RES	Electronic ID			Electronic ID			Electronic ID		
	14			15			16		
REMS/REMS2	Manufacturer ID	Device ID		Manufacturer ID	Device ID		Vlanufacturer ID	Device ID	
	C2	14		C2	15		C2	16	

(17) Enter Secured OTP (ENSO)

The ENSO instruction is for entering the additional 512-bit secured OTP mode. The additional 512-bit secured OTP is independent from main array, which may use to store unique serial number for system identifier. After entering the Secured OTP mode, and then follow standard read or program, procedure to read out the data or update data. The Secured OTP data cannot be updated again once it is lock-down.

The sequence of issuing ENSO instruction is: CS# goes low-> sending ENSO instruction to enter Secured OTP mode -> CS# goes high.

Please note that WRSR/WRSCUR commands are not acceptable during the access of secure OTP region, once security OTP is lock down, only read related commands are valid.

(18) Exit Secured OTP (EXSO)

The EXSO instruction is for exiting the additional 512-bit secured OTP mode.

The sequence of issuing EXSO instruction is: CS# goes low-> sending EXSO instruction to exit Secured OTP mode-> CS# goes high.

(19) Read Security Register (RDSCUR)

The RDSCUR instruction is for reading the value of Security Register bits. The Read Security Register can be read at any time (even in program/erase/write status register/write security register condition) and continuously.

The sequence of issuing RDSCUR instruction is : CS# goes low-> send ing RDSCUR instruction -> Security Register data out on SO-> CS# goes high.

The definition of the Security Register bits is as below:

Secured OTP Indicator bit. The Secured OTP indicator bit shows the chip is locked by factory before ex- factory or not. When it is "0", it indicates non- factory lock; "1" indicates factory- lock.

Lock-down Secured OTP (LDSO) bit. By writing WRSCUR instruction, the LDSO bit may be set to "1" for customer lock-down purpose. However, once the bit is set to "1" (lock-down), the LDSO bit and the 512-bit Secured OTP area cannot be update any more. While it is in 512-bit secured OTP mode, array access is not allowed.

Table 8. Security Register Definition

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
x	x	x	x	x	x	LDSO (indicate if lock-down	Secrured OTP indicator bit
reserved	reserved	reserved	reserved	reserved	reserved	0 = not lock-down 1 = lock-down (cannot program/erase OTP)	0 = non-factory lock 1 = factory lock
volatile bit	volatile bit	volatile bit	volatile bit	volatile bit	volatile bit	non-volatile bit	non-volatile bit

(20) Write Security Register (WRSCUR)

The WRSCUR instruction is for changing the values of Security Register Bits. Unlike write status register, the WREN instruction is not required before sending WRSCUR instruction. The WRSCUR instruction may change the values of bit1 (LDSO bit) for customer to lock-down the 512-bit Secured OTP area. Once the LDSO bit is set to "1", the Secured OTP area cannot be updated any more.

The sequence of issuing WRSCUR instruction is :CS# goes low-> sending WRSCUR instruction -> CS# goes high.

The CS# must go high exactly at the boundary; otherwise, the instruction will be rejected and not executed.

Setup 1208AA and 1G08AA Flash

Add the Macronix manufacture id to the define list in *include/linux/nand.h*

```
548  * NAND Flash Manufacturer ID Codes
549  */
550 #define NAND_MFR_TOSHIBA    0x98
551 #define NAND_MFR_SAMSUNG    0xec
552 #define NAND_MFR_FUJITSU    0x04
553 #define NAND_MFR_NATIONAL   0x8f
554 #define NAND_MFR_RENESAS    0x07
555 #define NAND_MFR_STMICRO    0x20
556 #define NAND_MFR_HYNIX      0xad
557 #define NAND_MFR_MICRON     0x2c
558 #define NAND_MFR_AMD        0x01
559 #define NAND_MFR_MACRONIX   0xc2
```

Insert the “name”, “device id” and “memory density” in the `nand_flash_ids` instance of `nand_flash_dev` structure in file *drivers/mtd/nand/nand_ids.c*.

Because the MX30LF1G08AA’s (1 Gigabit) information already exists, you only need to add the MX30LF1208AA (512 Megabit) information to this table.

```
24  struct nand_flash_dev nand_flash_ids[] = {

76      /*512 Megabit */
77      {"NAND 64MiB 1,8V 8-bit",   0xA2, 0,  64, 0, LP_OPTIONS},
78      {"NAND 64MiB 1,8V 8-bit",   0xA0, 0,  64, 0, LP_OPTIONS},
79      {"NAND 64MiB 3,3V 8-bit",   0xF2, 0,  64, 0, LP_OPTIONS},
80      {"NAND 64MiB 3,3V 8-bit",   0xD0, 0,  64, 0, LP_OPTIONS},
81      {"NAND 64MiB 3,3V 8-bit",   0xF0, 0,  64, 0, LP_OPTIONS},
82      {"NAND 64MiB 1,8V 16-bit",  0xB2, 0,  64, 0, LP_OPTIONS16},
83      {"NAND 64MiB 1,8V 16-bit",  0xB0, 0,  64, 0, LP_OPTIONS16},
84      {"NAND 64MiB 3,3V 16-bit",  0xC2, 0,  64, 0, LP_OPTIONS16},
85      {"NAND 64MiB 3,3V 16-bit",  0xC0, 0,  64, 0, LP_OPTIONS16},
```

And please list Macronix ID definition to struct `nand_manuf_ids` in the file *drivers/mtd/nand/nand_ids.c*.

```
170  struct nand_manufacturers nand_manuf_ids[] = {
171      {NAND_MFR_TOSHIBA, "Toshiba"},
172      {NAND_MFR_SAMSUNG, "Samsung"},
173      {NAND_MFR_FUJITSU, "Fujitsu"},
174      {NAND_MFR_NATIONAL, "National"},
175      {NAND_MFR_RENESAS, "Renesas"},
176      {NAND_MFR_STMICRO, "ST Micro"},
177      {NAND_MFR_HYNIX, "Hynix"},
178      {NAND_MFR_MICRON, "Micron"},
179      {NAND_MFR_AMD, "AMD"},
180      {NAND_MFR_MACRONIX, "Macronix"},
181      {0x0, "Unknown"}
182  };
```

Add information of the Macronix NAND device in the `nand_device_info_table_type_2` instance of the `nand_device_info` structure in the file *drivers/mtd/nand/nand_device_info.c*. This file only exists in i.MX28 kernel which is provided by Freescale. You can't find it in public release Linux source. It is used for the GPMI NAND controller.

There are totally 10 tables in `nand_device_info.c`. Why we choose type_2? The type 2, 7, 10 is used for SLC NAND flash (Single Level Cell) which always has smaller density, smaller write/erase unit and higher performance. Among them, type 7 is used for chips which have multi-plane and support simultaneously program command. And type 10 defines some special SLC NAND which has page size equal or larger than 4 Kbytes. So we define Macronix chip in `nand_device_info_table_type_2`.

The image below shows the information you must add for the 512Mbit MX30LF1208AA.

```
22  static struct nand_device_info nand_device_info_table_type_2[] __initdata = {
23      {
24      .end_of_table              = false,
25      .manufacturer_code         = 0xc2,
26      .device_code               = 0xf0,
27      .cell_technology           = NAND_DEVICE_CELL_TECH_SLC,
28      .chip_size_in_bytes        = 64LL*SZ_1M,
29      .block_size_in_pages       = 64,
30      .page_total_size_in_bytes  = 2*SZ_1K + 64,
31      .ecc_strength_in_bits      = 1,
32      .ecc_size_in_bytes         = 512,
33      .data_setup_in_ns          = 5,
34      .data_hold_in_ns           = 5,
35      .address_setup_in_ns       = 15,
36      .gpmi_sample_delay_in_ns   = 6,
37      .tREA_in_ns                = 20,
38      .tRLOH_in_ns               = -1,
39      .tRHOH_in_ns               = -1,
40      "MX30LF1208AA",
41      },
```

The image below shows the information you must add for the 1Gbit MX30LF1G08AA.

```
81      .end_of_table              = false,
82      .manufacturer_code         = 0xc2,
83      .device_code               = 0xf1,
84      .cell_technology           = NAND_DEVICE_CELL_TECH_SLC,
85      .chip_size_in_bytes        = 128LL*SZ_1M,
86      .block_size_in_pages       = 64,
87      .page_total_size_in_bytes  = 2*SZ_1K + 64,
88      .ecc_strength_in_bits      = 1,
89      .ecc_size_in_bytes         = 512,
90      .data_setup_in_ns          = 5,
91      .data_hold_in_ns           = 5,
92      .address_setup_in_ns       = 15,
93      .gpmi_sample_delay_in_ns   = 6,
94      .tREA_in_ns                = 20,
95      .tRLOH_in_ns               = -1,
96      .tRHOH_in_ns               = -1,
97      "MX30LF1G08AA",
98      },
```

Now setup the initialization function for the Macronix NAND flash devices. Here we build a new function naming `nand_device_info_fn_macronix`.

```
2152  static struct nand_device_info * __init nand_device_info_fn_macronix(const uint8_t id[])
2153  {
2154      /* SLC device only */
2155      /* Type 2 */
2156      return nand_device_info_search(nand_device_info_table_type_2,
2157                                     ID_GET_MFR_CODE(id), ID_GET_DEVICE_CODE(id));
2158  }
```

Make a new element in the `nand_device_mfr_directory` array and assign to .id variable the `NAND_MFR_MACRONIX` define and the `nand_device_info_fn_macronix` to the .fn variable.

```
2312  static struct nand_device_mfr_info  nand_device_mfr_directory[] __initdata = {
2313      {
2314      .id = NAND_MFR_MACRONIX,
2315      .fn = nand_device_info_fn_macronix,
2316      },
```

For other Linux versions, please refer to NAND driver patch in our website under the support area for more information.

Setup 2G28AB and 4G28AB N36 Flash

The fundamental modifications are the same as last section. We don't need to add chip information to `nand_flash_ids` table because they already exist. But we should add information to `nand_device_info_table_type_2` as following.

Macronix MX30LF2G28AB and MX30LF4G28AB support up to 112-byte spare area. But for hardware ECC compatibility consider, we suggest that if you don't need more than 64-byte spare area, please set `page_total_size_in_bytes` to value 2*SZ_1K + 64.

```
22  static struct nand_device_info nand_device_info_table_type_2[] __initdata = {
23      {
24      .end_of_table             = false,
25      .manufacturer_code        = 0xc2,
26      .device_code              = 0xda,
27      .cell_technology          = NAND_DEVICE_CELL_TECH_SLC,
28      .chip_size_in_bytes       = 256LL*SZ_1M,
29      .block_size_in_pages      = 64,
30      .page_total_size_in_bytes = 2*SZ_1K + 112,
31      .ecc_strength_in_bits     = 8,
32      .ecc_size_in_bytes        = 512,
33      .data_setup_in_ns         = 7,
34      .data_hold_in_ns          = 5,
35      .address_setup_in_ns      = 10,
36      .gpmi_sample_delay_in_ns  = 6,
37      .tREA_in_ns               = 16,
38      .tRLOH_in_ns              = 5,
39      .tRHOH_in_ns              = -1,
40      "MX30LF2G28AB",
41      },
42      {
43      .end_of_table             = false,
44      .manufacturer_code        = 0xc2,
45      .device_code              = 0xdc,
46      .cell_technology          = NAND_DEVICE_CELL_TECH_SLC,
47      .chip_size_in_bytes       = 512LL*SZ_1M,
48      .block_size_in_pages      = 64,
49      .page_total_size_in_bytes = 2*SZ_1K + 112,
50      .ecc_strength_in_bits     = 8,
51      .ecc_size_in_bytes        = 512,
52      .data_setup_in_ns         = 7,
53      .data_hold_in_ns          = 5,
54      .address_setup_in_ns      = 10,
55      .gpmi_sample_delay_in_ns  = 6,
56      .tREA_in_ns               = 16,
57      .tRLOH_in_ns              = 5,
58      .tRHOH_in_ns              = -1,
59      "MX30LF4G28AB",
60      },
```

BTW, actually, we should add above data to `nand_device_info_table_type_7` which is set for chip support multi-plane operation. But there is no difference between two types to our driver, so directly add device information to `nand_device_info_table_type_2` will be easier.

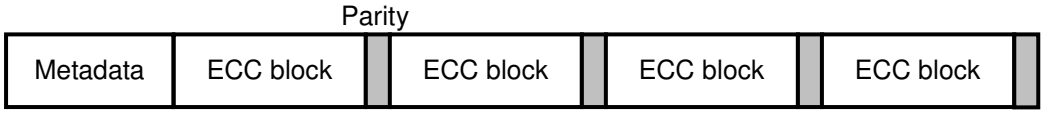
The i.MX28 default use 8-bit ECC with 2KB page NAND flash. You must change ECC strength in file *drivers/mtd/nand/gpmi-nfc/gpmi-nfc-hal-comon.c* instead of changing `ecc_strength_in_bits` in `nand_device_info_table_type_2` array.

You could refer to document “How to handle the spare-byte area of Macronix N36 NAND Flash” (*Technical Documents/Application Notes/SLC NAND/General Design-in Guidelines* on Macronix website), in that we’ll show you how to check your ECC strength.

```
147  int gpmi_nfc_set_geometry(struct gpmi_nfc_data *this) {
222
223      switch (physical->page_data_size_in_bytes) {
224      case 2048:
225          geometry->ecc_strength = 8;
226          break;
227      case 4096:
228          switch (physical->page_oob_size_in_bytes) {
229          case 128:
230              geometry->ecc_strength = 8;
231              break;
232          case 218:
233              geometry->ecc_strength = 16;
234              break;
235          }
236          break;
```

If you are going to use 112-byte spare area and already set `page_total_size_in_bytes` to 2*SZ_1K + 112, you may need to check hardware ECC setting in GPMI driver. BUT without modification can still work.

The GPMI driver has setup a space “metadata” used for store important data of file system. We suggest you to reset the “metadata size” and “ECC strength” for Macronix chip. The following formula will show you why we do in this way.



i.MX28 Hardware ECC layout in a page

```
M = Free spare area reserved for metadata
E0 = ECC strength of metadata (number of bits to be corrected per 512bytes)
E1 = ECC strength of user data
Macronix chip data: page_size = 2048, spare_area_size = 112
Formula:
E0 * 13 <= available_spare_area_bits - (E1 * (block_num) * 13)
   <= ((spare_area_size - M) * bits_per_byte) - (E1 * (page_size / 512) * 13)
   <= ((112 - M) * 8) - (E1 * (2048 / 512) * 13)
   <= 896 - M * 8 - E1 * 52
```

For example, if we just want to use 8-bit ECC for user data (E1 = 8). And assume metadata at least need 8-bit ECC (E0 = 8), then we can use up to 14-bit ECC (E1 = 14) for user data as we set metada_size to 8 (M = 8).

Other way, if we want to setup same ECC strength in user data and metadata (E0 = E1), then the max ECC we can use is (spare_area_bits / 65). And after we get ECC strength, we can get the max metadata_size we should set.

```
E <= available_spare_area_bits / 65
M <= (available_spare_area_bits - E * 65) / 8
```

You can change the file *drivers/mtd/nand/gpmi-nfc/gpmi-nfc-hal-comon.c* as following. The code will directly get the best ECC strength base on your `page_oob_size_in_bytes`, and the metadata size will be auto-reset.

```
147  int gpmi_nfc_set_geometry(struct gpmi_nfc_data *this)
148  {

172      /*
173       * protect area = 2K page + metadata = 2K/512 + 1 = 5
174       * so at least 5 * 13 = 65 bytes is needed for ECC parity in 2K page
175       * simple formula: ecc_strength <= (8 * spare_area_size) / 65
176       */
177      max_ecc = 8 * physical->page_oob_size_in_bytes / 65;
178      for (ecc = 20; ecc > max_ecc && ecc >= 2; ecc = ecc - 2);
179
180      geometry->metadata_size_in_bytes =
181          (physical->page_oob_size_in_bytes * 8 - (ecc * 65)) / 8;
```

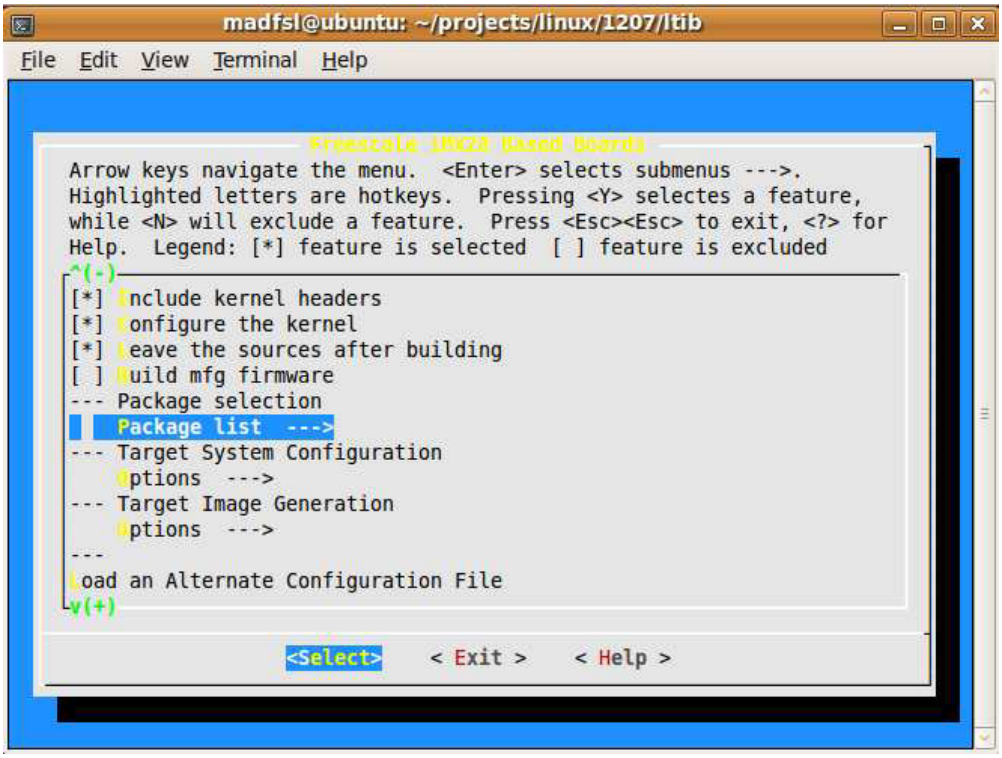
```
220     switch (physical->page_data_size_in_bytes) {
221     case 2048:
222         geometry->ecc_strength = 8;
223         break;
224     case 4096:
225         switch (physical->page_oob_size_in_bytes) {
226         case 128:
227             geometry->ecc_strength = 8;
228             break;
229         case 218:
230             geometry->ecc_strength = 16;
231             break;
232         }
233         break;
234     case 8192:
235         geometry->ecc_strength = 24;
236         /*
237          * ONFI/TOGGLE nand needs GF14, so re-culculate DMA page size.
238          * The ONFI nand must do the reculation,
239          * else it will fail in DMA.
240          */
241         if (is_ddr_nand(&this->device_info))
242             geometry->page_size_in_bytes =
243                 physical->page_data_size_in_bytes +
244                 geometry->metadata_size_in_bytes +
245                 (geometry->ecc_strength * 14 * 8 /
246                     geometry->ecc_chunk_count);
247         break;
248     }
249
250     geometry->ecc_strength = ecc;
251
```

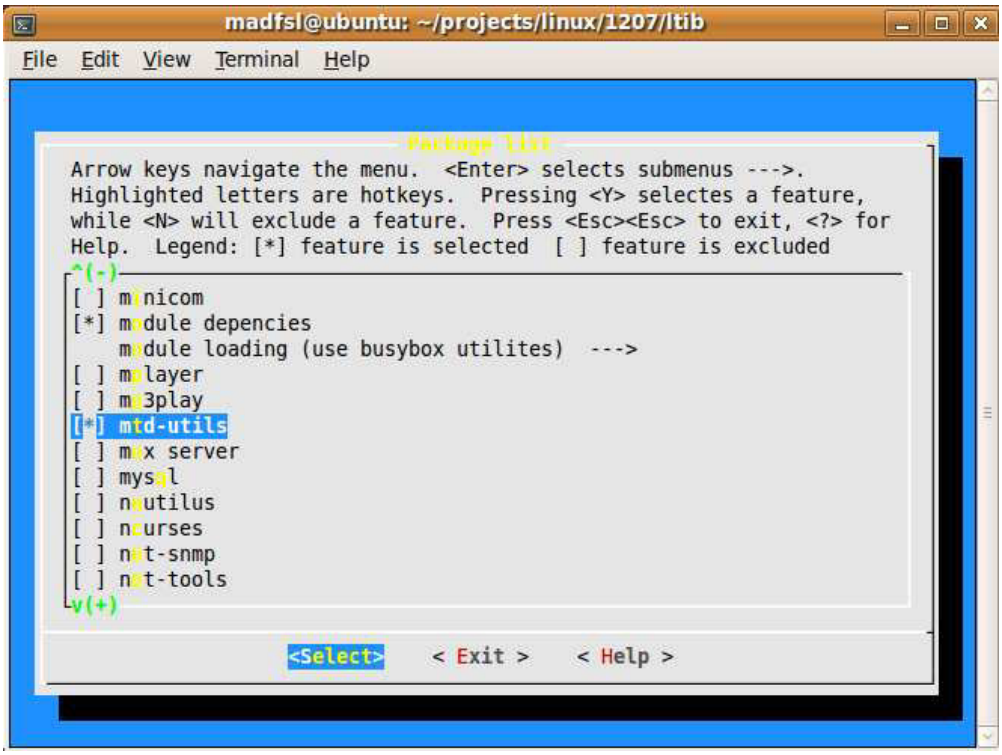
In case of Macronix MX30LF4G28AB, 12-bit hardware ECC will be set and the metadata size should be 14 bytes.

Linux Kernel Configuration

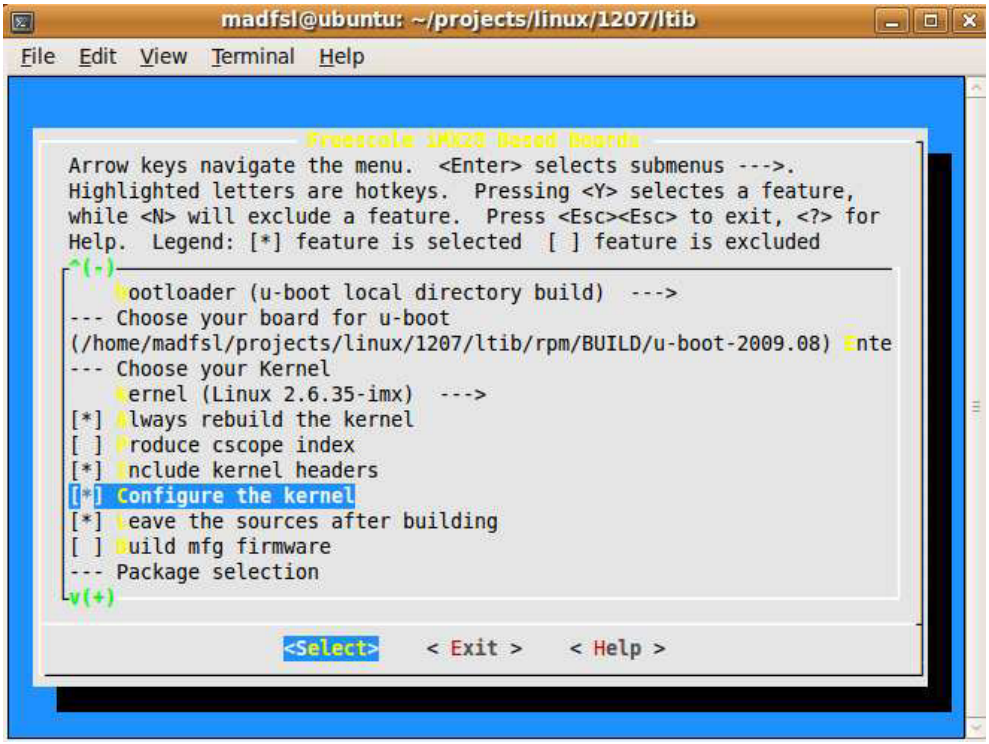
Run ltib script with argument “-m config” to configure the board and enable “mtd-utils”, which you can find it in “Package list”. The tool is useful in testing flash memory.

```
# ./ltib -m config
```





Choose “Configure the kernel” then exit and save.

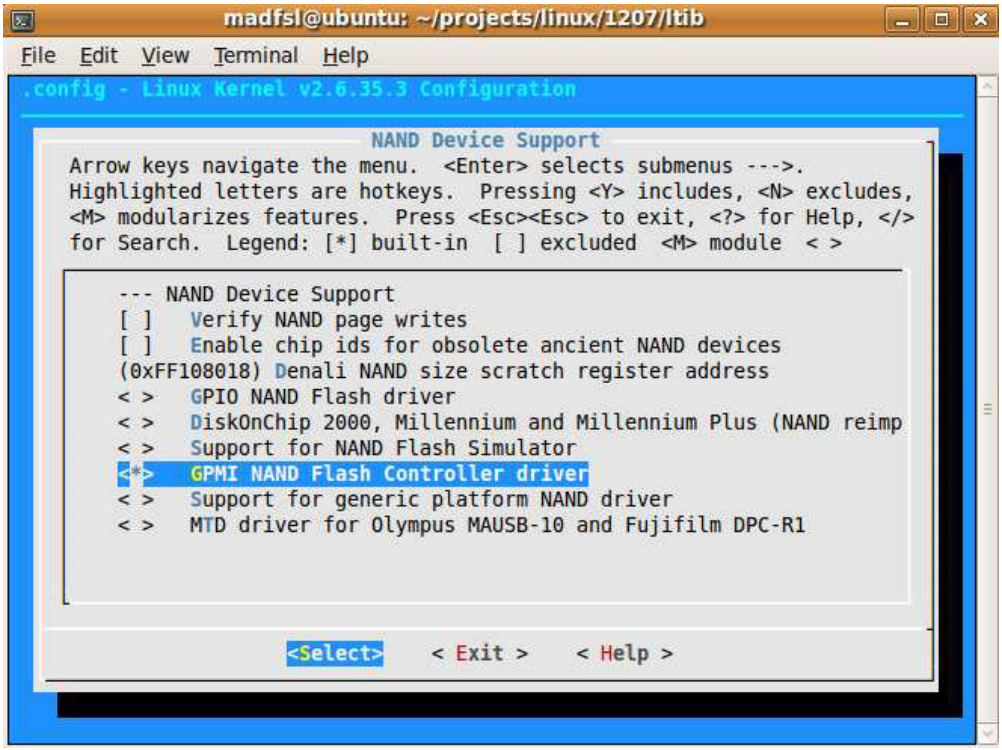


Run ltib to configure Linux kernel and rebuild kernel.

```
# ./ltib
```

In “menuconfig” window, you may need to select the following options for supporting i.MX28’s NAND controller.

```
<*> Device Drivers ->
    <*> Memory Technology Device (MTD) support ->
        <*> NAND Device Support ->
            <*> GPMI NAND Flash Controller driver
```



Then, you can follow i.MX28 setup steps to build kernel and rootfs to SD card.

```
# ./ltib -p boot_stream.spec -f
# umount /dev/sdc3
# ./mk_mx28_sd /dev/sdc  // sdc is your SD card device
```

Flash File Systems Test

Insert SD card boot i.MX28 from SD card with switch setting “1001”. You could check NAND device and GPMI controller’s working status with following command.

```
# cat /proc/mtd
dev:    size    erasesize    name
mtd0: 01400000 00020000 “gpmi-nfc-0-boot”
mtd1: 06c00000 00020000 “gpmi-nfc-0-general-use”
```

You could directly mount NAND device on “mtdblock” with ext2 file system. So how, “mtdblock” is a bad performance solution that is suitable to sequential access data, so we suggest you to try UBIFS.

```
# mkfs.ext2 /dev/mtdblock1
# mount -t ext2 /dev/mtdblock1 /mnt
# umount /mnt
```

Or you could mount NAND device with Journaling Flash File System (JFFS2).

```
# mount -t jffs2 /dev/mtdblock1 /mnt
# umount /mnt
```

Or you could mount with Unsorted Block Image File System (UBIFS).

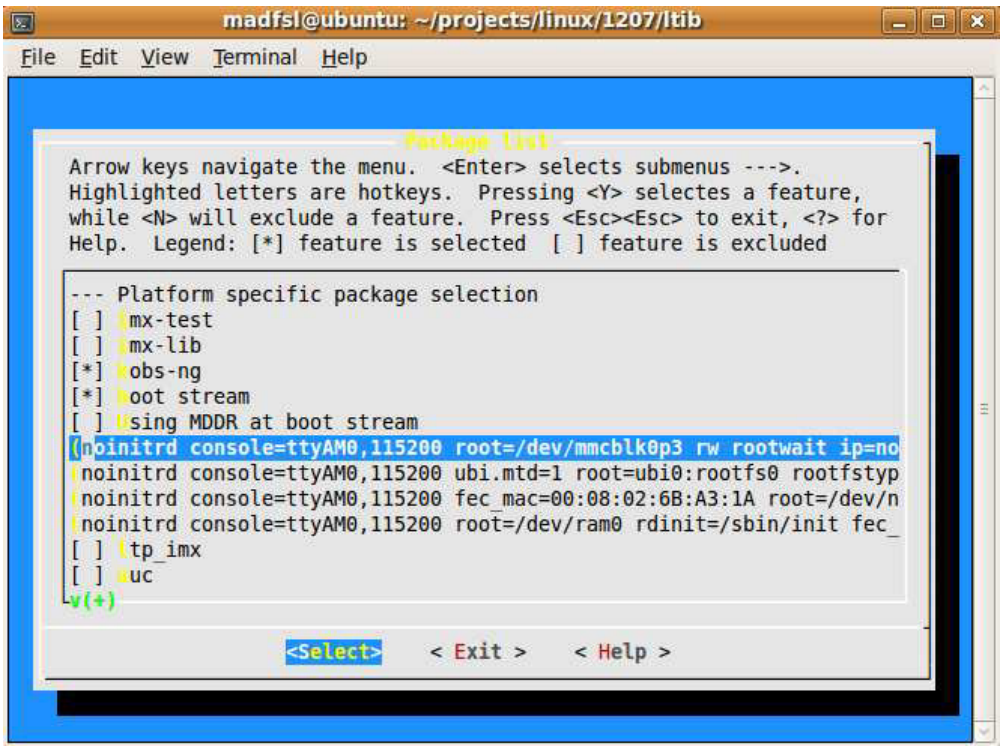
```
# flash_eraseall /dev/mtd1
# ubiattach /dev/ubi_ctrl -m 1
# ubimkvol /dev/ubi0 -N rootfs0 -s 100MiB
# mount -t ubifs ubi0:rootfs0 /mnt
  // or “mount -t ubifs ubi0_0 /mnt”
# umount /mnt
# ubirmvol /dev/ubi0 -n 0          // remove UBI volumn
# ubidetach /dev/ubi_ctrl -m 1     // un-link UBI manager
```

Boot Linux from NAND Flash

Configure the board with command "`./ltib -m config`" again. Select "Package list" -> "Boot stream", and retype the boot stream from the following (1) to (2), which means your root position is on MTD block device instead of MMC device (SD card). And the root file system is also need to be changed to flash file system such as UBIFS. The same procedure, if you want to boot with JFFS2 root file system, you'll need to try option (3).

(1) noinitrd console=ttyAM0, 115200 root=/dev/mmcblk0p3 rw rootwait ip=none gpmi
(2) noinitrd console=ttyAM0, 115200 ubi.mtd=1 root=ubi0:rootfs0 rootfstype=ubifs rw gpmi
(3) noinitrd console=ttyAM0, 115200 root=/dev/mtdblock1 rootfstype=jffs2 rw gpmi

Please refer to the following graph. After your setting, please save and exit.



Rebuild kernel with "`./ltib`" and "`./ltib -p boot_stream.spec -f`". Then copy the new kernel and root file system to SD card.

```
# cd ~/projects/linux/1008/ltib
# cp rootfs/boot/imx28_linux.sb /media/disk-1
# cp -rf rootfs tempfs
# rm -rf tempfs/boot              // no need boot directory
# tar -cf nandfs.tar tempfs/*
# cp nandfs.tar /media/disk-1     // copy rootfs to SD card
```

Boot from SD card and copy kernel and root file system to NAND device with UBIFS. These steps are similar in booting with JFFS2.

```
# flash_eraseall /dev/mtd0
# kobs-ng init imx28_linux.sb
  // copy kernel to NAND partition 0
# flash_eraseall /dev/mtd1
# ubiattach /dev/ubi_ctrl -m 1
# ubimkvol /dev/ubi0 -N rootfs0 -s 100MiB
# mount -t ubifs ubi0:rootfs0 /mnt
# tar -xf nandfs.tar -C /mnt   // copy rootfs to NAND
# unmount /mnt
# ubirmvol /dev/ubi0 -n 0
# ubidetach /dev/ubi_ctrl -m 1
```

Halt system and change switch to “0100”, then you can boot from NAND. If you want to boot from NAND with JFFS2 root file system, you can replace above code to the followings.

```
# flash_eraseall /dev/mtd0
# kobs-ng init imx28_linux.sb
# flash_eraseall /dev/mtd1
# mount -t jffs2 /mnt
# tar -xf nandfs.tar -C /mnt
# unmount /mnt
```