Hands-on Exercise: Interfacing CY14B101Q2A-SXI nvSRAM with IMXRT1050-EVKB

This exercise shows how to connect and use the **CY14B101Q2A-SXI** 1 Mbit SPI nvSRAM as an external memory for the **EVKB-IMXRT1050** board. You will:

- Wire the nvSRAM to the EVKB via Arduino SPI pins.
- Create an MCUXpresso project based on the **lpspi_loopback** SDK example.
- Implement SPI routines to **write and read** nvSRAM.
- Implement an **avionics-style configuration/health record** stored in nvSRAM.

---

1. Hardware hookup – CY14B101Q2A-SXI to IMXRT1050-EVKB

We will use the **Arduino SPI header** on the EVKB.

EVKB Arduino SPI Pins (LPSPI1)

- D10 → GPIO_SD_B0_01 → SPI_SS → LPSPI1_PCS0 (chip select)
- D11 → GPIO_SD_B0_02 → SPI_SOUT (MOSI) → LPSPI1_SDO
- D12 → GPIO_SD_B0_03 → SPI_SIN (MISO) → LPSPI1_SDI
- D13 → GPIO_SD_B0_00 → SPI_CLK → LPSPI1_SCK

These are already configured for **LPSPI1** in the SDK `lpspi_loopback` example.

CY14B101Q2A-SXI Overview

- Capacity: 1 Mbit = 128K × 8 (0x20000 bytes).
- Address space: 0x00000 – 0x1FFFF.
- SPI commands:
  - READ (0x03) / WRITE (0x02) with **3-byte address** (A16..A0).
  - WREN (0x06) **must** be issued before any write.
  - Other commands: RDSR, WRSR, STORE, RECALL, etc.
- Package: 8-pin SOIC with AutoStore (Q2A variant), using **VCAP** instead of WP.

Pin Connections

Assuming your CY14B101Q2A-SXI is on an SOIC-8 breakout, connect:

CY14B101Q2A Pin	Function	Connect to EVKB	Notes
1	HOLD#	3.3V	Tie HIGH to avoid pausing SPI transfers.

CY14B101Q2A Pin	Function	Connect to EVKB	Notes
2	SCK	Arduino D13	SPI clock (LPSPI1_SCK).
3	SI (MOSI)	Arduino D11	Master output to nvSRAM input.
4	VSS	GND	Common ground.
5	SO (MISO)	Arduino D12	nvSRAM output to master input.
6	CS#	Arduino D10	Chip select (LPSPI1_PCS0).
7	VCC	3.3V	Supply (2.7–3.6 V).
8	VCAP	0.1 µF to GND	Required for AutoStore.

Additional notes:

- Keep SPI signals at **3.3V**.
- Use short wires; twist with ground where possible for signal integrity.
- You can duplicate this wiring on multiple EVKBs for multi-channel experiments.

---

2. MCUXpresso Project Setup (using LPSPI example)

We reuse the **lpspi_loopback** example from the EVKB-IMXRT1050 SDK so all the pin mux and clocks for LPSPI1 are already configured.

Step 1 – Ensure SDK is installed

Install or verify the SDK for **EVKB-IMXRT1050** (matching your `SDK_25_06_00_EVKB-IMXRT1050` ZIP) in MCUXpresso.

Step 2 – Import the LPSPI loopback example

1. Open **MCUXpresso IDE**.
2. In the **Quickstart Panel**, click **Import SDK example(s)**.
3. Select board: **EVKB-IMXRT1050**.
4. From the examples list choose:
   - `demo_apps → lpspi_loopback`
5. Finish the wizard. A project like `evkbimxrt1050_lpspi_loopback` appears.

Step 3 – Build and test the stock example

- Build the project to ensure it compiles.
- Optionally run it once; it performs an internal LPSPI1 loopback test.

Step 4 – Replace the main source file

- Open `source/lpspi_loopback.c` in the project.
- **Replace the entire file content** with the code from section 3 below.
- Do not modify:
  - `app.h` (contains `BOARD_EEPROM_LPSPI_BASEADDR`, PCS macros, clock frequency).
  - `pin_mux.c` (configures LPSPI1 pins on GPIO_SD_B0_00..03).
  - `board.c`, `clock_config.c`, etc.

Step 5 – Rebuild

- Build the project again with the new source.

Step 6 – Debug configuration

1. Connect the EVKB to your PC via the **OpenSDA** USB connector.
2. Use the default **CMSIS-DAP** debug configuration that MCUXpresso creates.
3. Click the **Debug** button (green bug icon) to flash and run.

Step 7 – Serial output

- The example uses **PRINTF** via the board's debug UART (OpenSDA virtual COM).
- Open a terminal on the OpenSDA COM port:
  - Baud rate: **115200**, 8 data bits, no parity, 1 stop bit (115200 8-N-1).

---

3. C Code – CY14B101Q2A nvSRAM SPI Demo

Copy-paste this code into `lpspi_loopback.c` in the imported example project.

```c
/*
 * NVSRAM (CY14B101Q2A-SXI) SPI interface demo for EVKB-IMXRT1050
 *
 * This file is intended to replace lpspi_loopback.c in the
 * 'lpspi_loopback' MCUXpresso SDK example project.
 *
 * It reuses:
 *   - BOARD_InitHardware() from board.c
 *   - LPSPI1 pin mux and clock setup from pin_mux.c / clock_config.c
 *   - app.h macros for LPSPI1 base address and PCS selection
 */

#include "fsl_device_registers.h"
#include "fsl_lpspi.h"
#include "fsl_debug_console.h"
#include "board.h"
#include "app.h"
```

```
#include <string.h>

/***************************************************************************
***
 * Definitions

***************************************************************************
*/

/* CY14B101Q2A-SXI command opcodes */
#define NVSRAM_CMD_WREN     (0x06u) /* Write enable */
#define NVSRAM_CMD_WRDI     (0x04u) /* Write disable */
#define NVSRAM_CMD_RDSR     (0x05u) /* Read status register */
#define NVSRAM_CMD_WRSR     (0x01u) /* Write status register */
#define NVSRAM_CMD_READ     (0x03u) /* Read memory array */
#define NVSRAM_CMD_WRITE    (0x02u) /* Write memory array */
#define NVSRAM_CMD_FSTRD    (0x0Bu) /* Fast read */
#define NVSRAM_CMD_STORE    (0x3Cu) /* Software STORE to non-volatile */
#define NVSRAM_CMD_RECALL   (0x60u) /* Software RECALL from non-volatile */
#define NVSRAM_CMD_ASENB    (0x59u) /* AutoStore enable */
#define NVSRAM_CMD_ASDISB   (0x19u) /* AutoStore disable */
#define NVSRAM_CMD_SLEEP    (0xB9u) /* Enter low power / STORE + sleep */

/* 1 Mbit = 128 K x 8 = 0x20000 bytes */
#define NVSRAM_SIZE_BYTES   (0x20000u)
#define NVSRAM_MAX_XFER     (64u)   /* demo helper: max payload bytes per
call */

/* Simple avionics configuration record stored in nvSRAM */
typedef struct
{
    uint32_t magic;           /* 0xA5A5A5A5 indicates valid struct */
    uint32_t firmwareVersion; /* arbitrary firmware version */
    uint32_t flightHours;     /* simulated accumulated flight hours */
    uint16_t lastFaultCode;   /* simulated last fault code */
    uint16_t reserved;
    uint32_t checksum;        /* simple additive checksum over first 5 fields
*/
} avionics_config_t;

/***************************************************************************
***
 * Prototypes

***************************************************************************
*/

/* Reuse the same LPSPI1 initialization style as lpspi_loopback demo */
```

```
static void LPSPI_Init(void);

/* Low-level transfer helper */
static status_t NVSRAM_LpspiTransfer(uint8_t *txData,
                                     uint8_t *rxData,
                                     size_t dataSize);

/* nvSRAM protocol helpers */
static status_t NVSRAM_WriteEnable(void);
static status_t NVSRAM_WriteBytes(uint32_t address,
                                  const uint8_t *data,
                                  size_t length);
static status_t NVSRAM_ReadBytes(uint32_t address,
                                 uint8_t *data,
                                 size_t length);
static status_t NVSRAM_ReadStatus(uint8_t *status);

/* Demo helpers */
static void run_generic_pattern_test(void);
static void run_avionics_use_case_demo(void);
static uint32_t simple_checksum32(const uint32_t *data, size_t wordCount);

/*****************************************************************************
***
 * Code

*****************************************************************************
*/

static void LPSPI_Init(void)
{
    lpspi_master_config_t masterConfig;

    /* Master config - based on lpspi_loopback demo, baudrate at 500 kHz
     * which is well below the 40 MHz limit of standard READ/WRITE opcodes.
*/
    LPSPI_MasterGetDefaultConfig(&masterConfig);
    masterConfig.baudRate                   = 500000U;
    masterConfig.bitsPerFrame               = 8;
    masterConfig.cpol                       =
kLPSPI_ClockPolarityActiveHigh;
    masterConfig.cpha                       = kLPSPI_ClockPhaseFirstEdge;
    masterConfig.direction                  = kLPSPI_MsbFirst;
    masterConfig.whichPcs                   = BOARD_LPSPI_PCS_FOR_INIT;
    masterConfig.pcsActiveHighOrLow         = kLPSPI_PcsActiveLow;
    masterConfig.pinCfg                     = kLPSPI_SdiInSdoOut;
    masterConfig.dataOutConfig              = kLpspiDataOutTristate;
    masterConfig.pcsToSckDelayInNanoSec     = 1000000000 /
masterConfig.baudRate;
```

```
    masterConfig.lastSckToPcsDelayInNanoSec     = 1000000000 /
masterConfig.baudRate;
    masterConfig.betweenTransferDelayInNanoSec = 1000000000 /
masterConfig.baudRate;
    masterConfig.enableInputDelay              = false;

    LPSPI_MasterInit(BOARD_EEPROM_LPSPI_BASEADDR, &masterConfig,
BOARD_LPSPI_CLK_FREQ);
}

/* Low-level LPSPI blocking transfer wrapper */
static status_t NVSRAM_LpspiTransfer(uint8_t *txData,
                                     uint8_t *rxData,
                                     size_t dataSize)
{
    lpspi_transfer_t xfer;

    memset(&xfer, 0, sizeof(xfer));
    xfer.txData      = txData;
    xfer.rxData      = rxData;
    xfer.dataSize    = dataSize;
    xfer.configFlags = BOARD_LPSPI_PCS_FOR_TRANSFER |
kLPSPI_MasterPcsContinuous;

    return LPSPI_MasterTransferBlocking(BOARD_EEPROM_LPSPI_BASEADDR, &xfer);
}

/* Send WREN (Write Enable) command */
static status_t NVSRAM_WriteEnable(void)
{
    uint8_t cmd = NVSRAM_CMD_WREN;

    return NVSRAM_LpspiTransfer(&cmd, NULL, 1u);
}

/* Read Status Register (RDSR) */
static status_t NVSRAM_ReadStatus(uint8_t *status)
{
    uint8_t tx[2] = { NVSRAM_CMD_RDSR, 0xFFu };
    uint8_t rx[2] = { 0 };

    status_t result = NVSRAM_LpspiTransfer(tx, rx, sizeof(tx));
    if (result == kStatus_Success)
    {
        *status = rx[1]; /* first received byte is dummy, second is status */
    }
    return result;
}
```

```c
/* Write up to NVSRAM_MAX_XFER bytes to nvSRAM */
static status_t NVSRAM_WriteBytes(uint32_t address,
                                  const uint8_t *data,
                                  size_t length)
{
    status_t result;
    uint8_t  localBuf[4 + NVSRAM_MAX_XFER];

    if ((address + length) > NVSRAM_SIZE_BYTES)
    {
        return kStatus_InvalidArgument;
    }

    if (length > NVSRAM_MAX_XFER)
    {
        return kStatus_InvalidArgument;
    }

    /* Write Enable must precede any WRITE instruction */
    result = NVSRAM_WriteEnable();
    if (result != kStatus_Success)
    {
        return result;
    }

    /* Command + 3-byte address (A16..A0) */
    localBuf[0] = NVSRAM_CMD_WRITE;
    localBuf[1] = (uint8_t)((address >> 16) & 0x01u); /* A16 in bit0 */
    localBuf[2] = (uint8_t)((address >> 8) & 0xFFu);  /* A15..A8 */
    localBuf[3] = (uint8_t)(address & 0xFFu);         /* A7..A0 */

    memcpy(&localBuf[4], data, length);

    return NVSRAM_LpspiTransfer(localBuf, NULL, 4u + length);
}

/* Read up to NVSRAM_MAX_XFER bytes from nvSRAM */
static status_t NVSRAM_ReadBytes(uint32_t address,
                                 uint8_t *data,
                                 size_t length)
{
    status_t result;
    uint8_t  txBuf[4 + NVSRAM_MAX_XFER];
    uint8_t  rxBuf[4 + NVSRAM_MAX_XFER];

    if ((address + length) > NVSRAM_SIZE_BYTES)
    {
        return kStatus_InvalidArgument;
    }
    if (length > NVSRAM_MAX_XFER)
    {
        return kStatus_InvalidArgument;
    }

    /* Command + 3-byte address */
    txBuf[0] = NVSRAM_CMD_READ;
    txBuf[1] = (uint8_t)((address >> 16) & 0x01u);
    txBuf[2] = (uint8_t)((address >> 8) & 0xFFu);
    txBuf[3] = (uint8_t)(address & 0xFFu);

    memset(&txBuf[4], 0xFFu, length); /* dummy bytes to clock data out */

    result = NVSRAM_LpspiTransfer(txBuf, rxBuf, 4u + length);
    if (result == kStatus_Success)
    {
        memcpy(data, &rxBuf[4], length);
    }
    return result;
}

/* Simple additive checksum over wordCount 32-bit words */
static uint32_t simple_checksum32(const uint32_t *data, size_t wordCount)
{
    uint32_t sum = 0;
    size_t   i;

    for (i = 0; i < wordCount; i++)
    {
        sum += data[i];
    }
    return sum;
}

/* Generic pattern write/read test */
static void run_generic_pattern_test(void)
{
    status_t result;
    uint8_t  txPattern[16];
    uint8_t  rxPattern[16];
    uint32_t testAddress = 0x000010u;
    uint32_t i;

    PRINTF("\r\n[GENERIC] nvSRAM pattern test starting...\r\n");

    for (i = 0; i < sizeof(txPattern); i++)
    {
        txPattern[i] = (uint8_t)(i * 3u + 0x55u);
        rxPattern[i] = 0u;
    }

    PRINTF("[GENERIC] Writing %d bytes at address 0x%05X...\r\n",
           sizeof(txPattern), testAddress);
    result = NVSRAM_WriteBytes(testAddress, txPattern, sizeof(txPattern));
    if (result != kStatus_Success)
    {
        PRINTF("[GENERIC] Write failed, status = %d\r\n", result);
        return;
    }

    PRINTF("[GENERIC] Reading back...\r\n");
    result = NVSRAM_ReadBytes(testAddress, rxPattern, sizeof(rxPattern));
    if (result != kStatus_Success)
    {
        PRINTF("[GENERIC] Read failed, status = %d\r\n", result);
        return;
    }

    for (i = 0; i < sizeof(txPattern); i++)
    {
        if (txPattern[i] != rxPattern[i])
        {
            PRINTF("[GENERIC] MISMATCH at index %d: wrote 0x%02X, read
0x%02X\r\n",
                   i, txPattern[i], rxPattern[i]);
            PRINTF("[GENERIC] Pattern test FAILED.\r\n");
            return;
        }
    }

    PRINTF("[GENERIC] Pattern test PASSED.\r\n");
}

/* Avionics-style use case: store configuration/health snapshot */
static void run_avionics_use_case_demo(void)
{
    status_t          result;
    avionics_config_t cfg;
    avionics_config_t cfgReadback;
    uint32_t          baseAddress = 0x001000u; /* arbitrary offset in nvSRAM
*/

    PRINTF("\r\n[AVIONICS] Storing configuration snapshot into
nvSRAM...\r\n");

    memset(&cfg, 0, sizeof(cfg));
    cfg.magic           = 0xA5A5A5A5u;
```

```c
    cfg.firmwareVersion = 0x00010002u; /* v1.2 for example */
    cfg.flightHours     = 1234u;       /* pretend we accumulated 1234 hours
*/
    cfg.lastFaultCode   = 0x0042u;     /* pretend last fault code */
    cfg.reserved        = 0u;
    cfg.checksum        = simple_checksum32((const uint32_t *)&cfg,
                                           (sizeof(cfg) - sizeof(uint32_t))
/ sizeof(uint32_t));

    /* Write struct */
    result = NVSRAM_WriteBytes(baseAddress, (const uint8_t *)&cfg,
sizeof(cfg));
    if (result != kStatus_Success)
    {
        PRINTF("[AVIONICS] Write failed, status = %d\r\n", result);
        return;
    }

    /* Read back */
    memset(&cfgReadback, 0, sizeof(cfgReadback));
    result = NVSRAM_ReadBytes(baseAddress, (uint8_t *)&cfgReadback,
sizeof(cfgReadback));
    if (result != kStatus_Success)
    {
        PRINTF("[AVIONICS] Read failed, status = %d\r\n", result);
        return;
    }

    /* Verify */
    if (cfgReadback.magic != 0xA5A5A5A5u)
    {
        PRINTF("[AVIONICS] Invalid magic, got 0x%08X\r\n",
cfgReadback.magic);
        return;
    }

    {
        uint32_t expectedChecksum =
            simple_checksum32((const uint32_t *)&cfgReadback,
                              (sizeof(cfgReadback) - sizeof(uint32_t)) /
sizeof(uint32_t));
        if (expectedChecksum != cfgReadback.checksum)
        {
            PRINTF("[AVIONICS] Checksum mismatch! expected 0x%08X, stored
0x%08X\r\n",
                   expectedChecksum, cfgReadback.checksum);
            return;
        }
    }
    PRINTF("[AVIONICS] Config snapshot verified OK.\r\n");
    PRINTF("[AVIONICS] firmwareVersion = 0x%08X, flightHours = %u, lastFault
= 0x%04X\r\n",
           cfgReadback.firmwareVersion,
           cfgReadback.flightHours,
           cfgReadback.lastFaultCode);

    PRINTF("[AVIONICS] To test non-volatility, power-cycle the board and run
this demo again.\r\n");
    PRINTF("          The same values should be read back from nvSRAM.\r\n");
}

/* Main entry */
int main(void)
{
    /* Init board hardware (clocks, pins, debug console, etc.). */
    BOARD_InitHardware();

    PRINTF("\r\n*** EVKB-IMXRT1050 + CY14B101Q2A-SXI nvSRAM Demo ***\r\n");

    LPSPI_Init();

    /* Run simple generic pattern test */
    run_generic_pattern_test();

    /* Run avionics-style use-case demo */
    run_avionics_use_case_demo();

    PRINTF("\r\nDemo complete. You can reset the board to repeat.\r\n");

    while (1)
    {
        __NOP();
    }
}
```

---

4. How to Run & Expected Output

1. Power up the EVKB and ensure the nvSRAM wiring is correct.
2. Start a debug session from MCUXpresso (using CMSIS-DAP).
3. Open your terminal (or use the MCUXpresso console) at 115200 8-N-1.
4. Reset/run the program.

You should see output similar to:

```
*** EVKB-IMXRT1050 + CY14B101Q2A-SXI nvSRAM Demo ***

[GENERIC] nvSRAM pattern test starting...
[GENERIC] Writing 16 bytes at address 0x00010...
[GENERIC] Reading back...
[GENERIC] Pattern test PASSED.

[AVIONICS] Storing configuration snapshot into nvSRAM...
[AVIONICS] Config snapshot verified OK.
[AVIONICS] firmwareVersion = 0x00010002, flightHours = 1234, lastFault =
0x0042

Demo complete. You can reset the board to repeat.
```

Non-volatility demonstration

1. After a successful run, **power off** the EVKB (switch off or unplug USB).
2. Wait a few seconds.
3. Power it on again and run the demo.
4. The same `firmwareVersion`, `flightHours`, and `lastFaultCode` are read back, showing that nvSRAM preserved the data across the power cycle.

---

5. Extensions (with 4 boards)

With four EVKB boards and four nvSRAM devices, you can simulate more advanced avionics scenarios:

1. **Redundant channels**
   - Each board represents an independent flight-control channel.
   - Each channel stores its own configuration/health record into its nvSRAM.
   - On boot, each channel validates the record (magic + checksum) and decides whether it is healthy.
2. **Flight counters and maintenance logging**
   - On each simulated flight cycle, increment `flightHours` or add a `flightCount` field.
   - Store fault occurrences, last reset reason, or self-test results in nvSRAM.
3. **Configuration updates**
   - Add a command interface (UART menu) to change configuration values.
   - After changing parameters, recompute checksum and write the updated struct to nvSRAM.
4. **STORE/RECALL and status register**
   - Extend the driver with explicit `STORE` and `RECALL` commands.
   - Read the status register after STORE to confirm operation.

- Compare behaviour with and without AutoStore enabled.