Implementing Microwire Protocol by PSIO

Application Note for 32-bit NuMicro® Family

Document Information

Abstract	Describe how to implement Microwire protocol by PSIO.
Apply to	NuMicro® M251/M252 Series

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

Table of Contents

1 Introduction

PSIO (Programmable Serial I/O) provides an easy way to receive and transmit a variety of serial transmissions such as Microwire, HDQ, DMX512, 1-wire, IR, PS/2, Wiegand, and LED.

Microwire is a subset of SPI. This article describes how to implement the Microwire protocol using PSIO and access the EEPROM using the Microwire protocol; it contains a basic introduction to PSIO and Microwire, the sample program description, and the results of the run.

1.1 Atmel Microwire Serial EEPROM

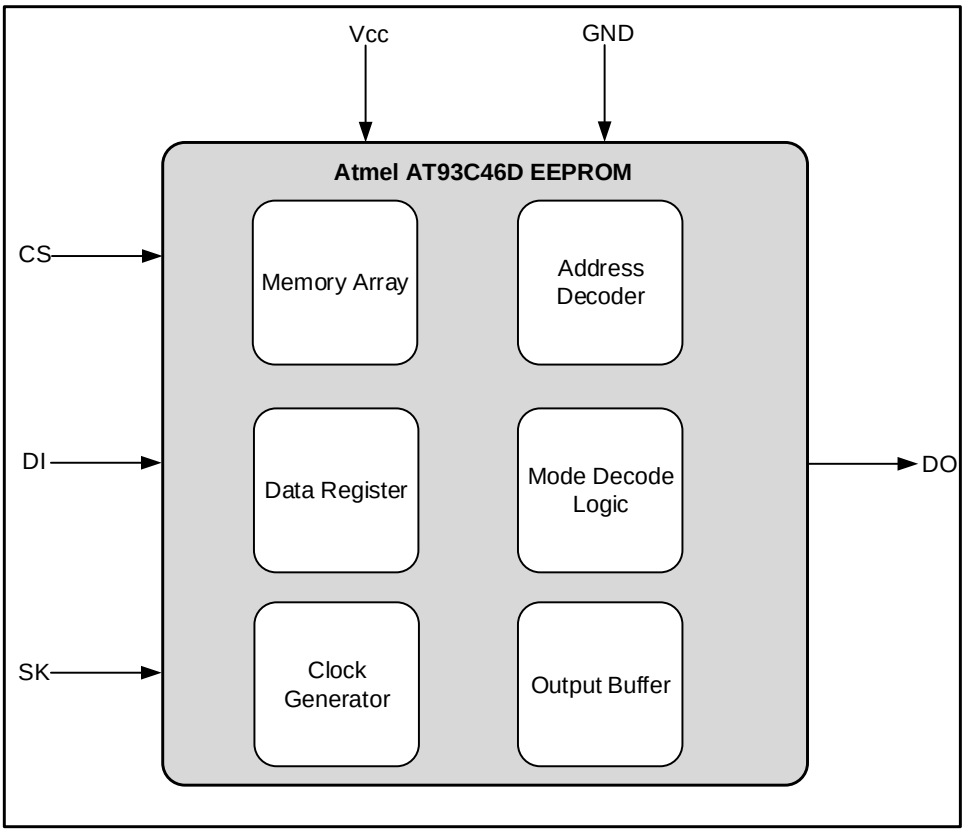

Figure 1-1 Atmel AT93C46D EEPROM

The Atmel AT93C46D is a 1024-bit EEPROM[1] that access by CS (Chip Select), SK (Shift Clock), DI (Data In), and DO (Data Out). It operates from 1.8 V to 5.5 V, and the clock rate can reach 2 MHz in a working environment with a voltage of 5 V. Figure 1-1 shows the system block diagram of the AT93C46D.

1.2 Sample System

Figure 1-2 is the system architecture diagram of this example, which implements Microwire protocol using PSIO of M251/M252, and read/write Atmel AT93C46D EEPROM. The wiring method is M251/M252 PB.15 is connected to the AT93C46D CS pin, PB.14 is connected to

the SK pin, PC.2 (PSIO output) is connected to the DI pin, and PC.3 (PSIO input) is connected to the DO pin.

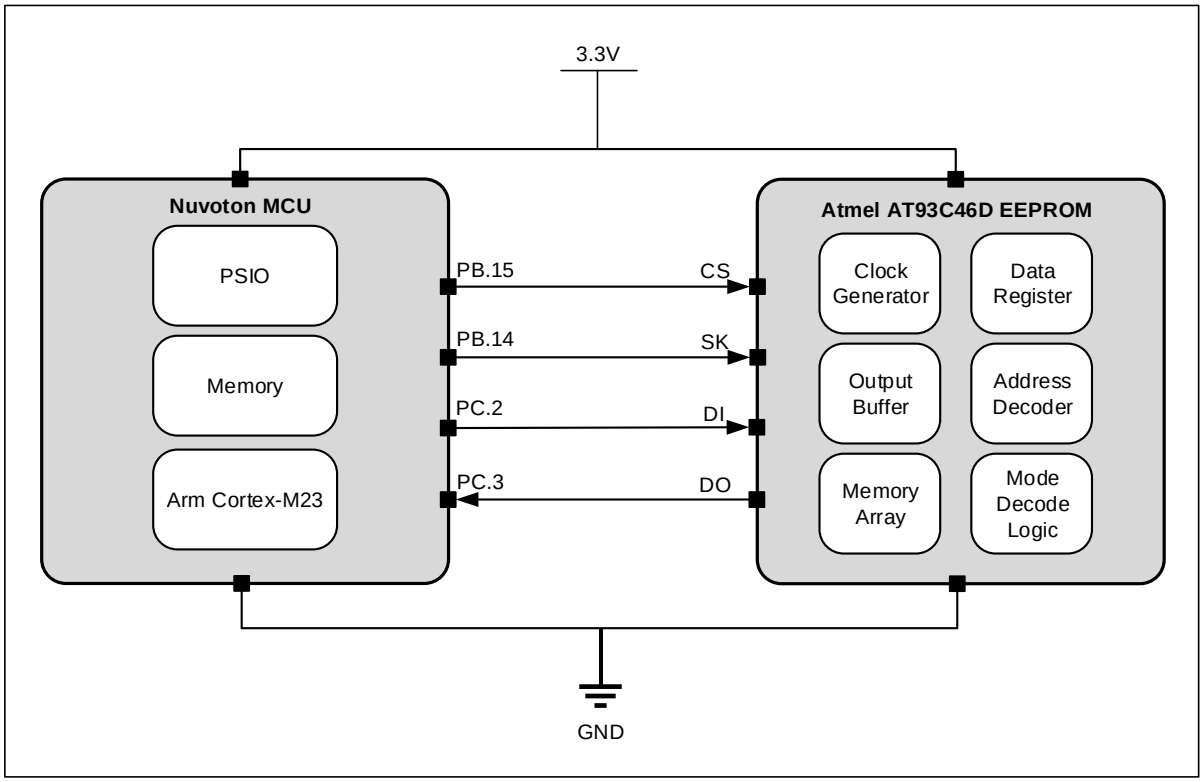

Figure 1-2 System Architecture

1.3 Introduction to PSIO

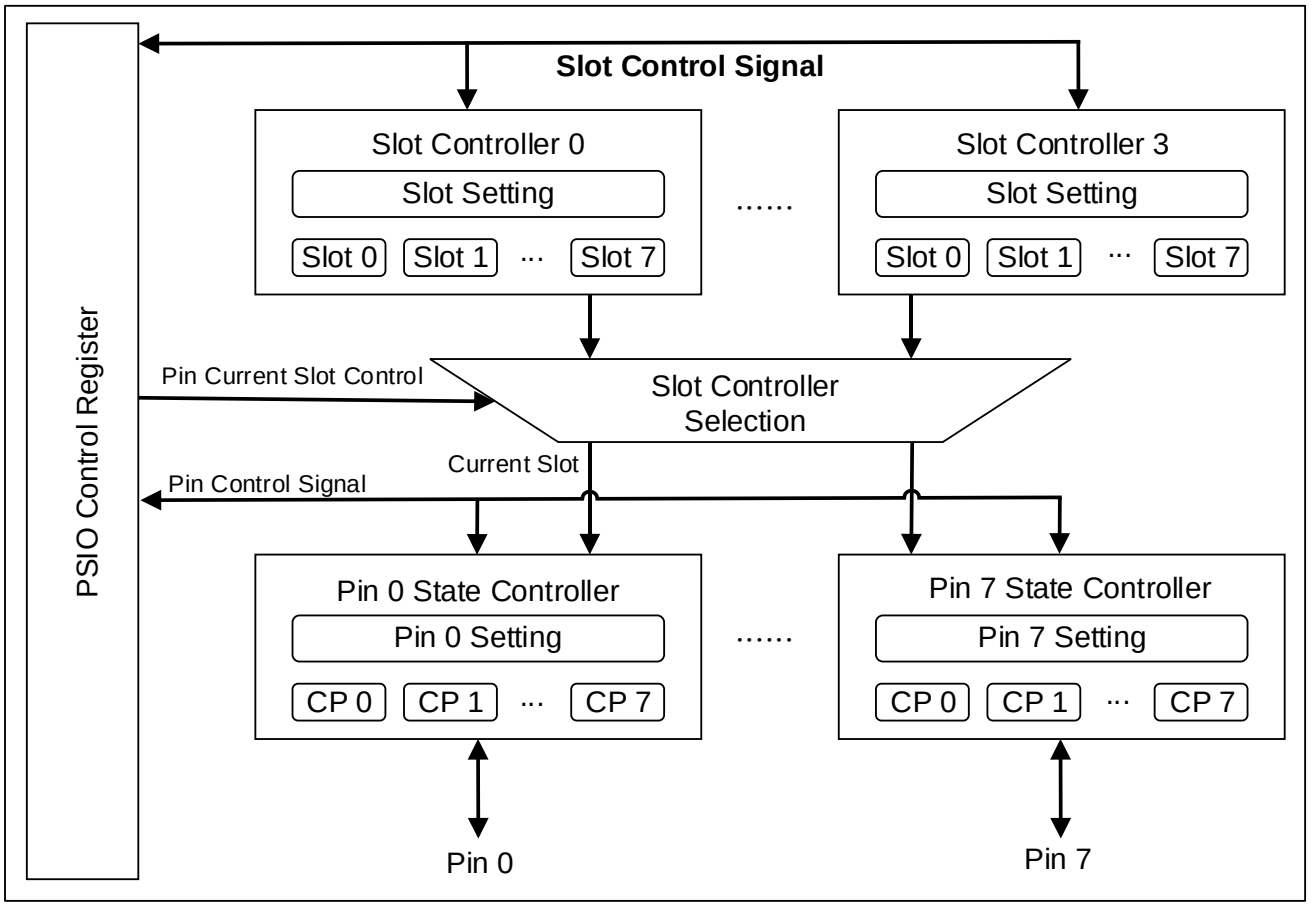

Figure 1-3 PSIO Block Diagram

The PSIO has 4 slot controllers and 8 pins. The slot controller is the timing control module. Each pin can select the corresponding slot controller to control its timing and complete the pin action at the corresponding timing such as input, output and other actions, as shown in Figure 1-3

1.3.1 Slot Controller

There are 8 slots in each slot controller for counting. The slot can be set to 0~15 engine clocks, and slot 0~slot 7 are executed sequentially. The method of triggering the slot controller to start counting can be divided into four types, software trigger, rising edge trigger, falling edge trigger and rising edge or falling edge trigger. In addition, three kinds of slot repeat counting modes are also provided, which are general repeat counting mode, general repeat counting with infinite loop mode and full repeat counting mode. The PSIO can expand the diversity of slot counts through these repeat counting modes.

1.3.2 Pin State Controller

Each Pin has 8 check points, and each check point can set Pin action as output (output low, output high, output from buffer, output toggle), input (input data buffer, input status, input

status record and update). Then set the check point to correspond to the slot. Through the above settings, when the slot starts counting in sequence, the Pin action will also change, and the state of pin will reach the expected according to timing.

1.3.3 Data Buffer Width and Depth

When the Pin action is set to be stored in the data buffer or read from the data buffer, the width and depth of the data buffer must be set to accommodate different bits of data.

The Data buffer is a 32-bit register, and the width can be set 1-bit to 32-bit. When the input or output data reaches the width setting value, the status of the PSIO Transfer Status Register will be marked full or empty. Depth is set to make the data buffer more efficient. The data buffer register has 32-bit. When the width is set to 1-bit to 8-bit, it can actually accommodate 4 data, depth can be set to 1 to 4, and so on, when the width is set to 9-bit to 16-bit, it can hold 2 data, and the depth can be set to 1 to 2. When the width is set to 17-bit to 32-bit, only one data can be accommodated, and the depth can only be set to 1, as shown in Figure 1-4.

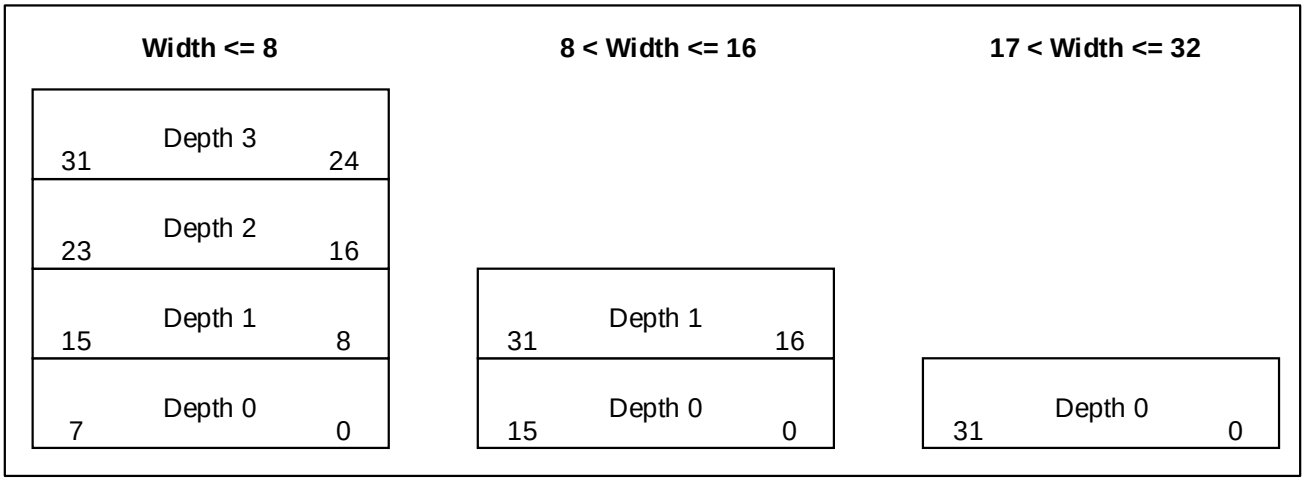

Figure 1-4 Data Buffer Width and Depth

1.4 Microwire Protocol

Microwire is a full-duplex, 3-wire communication interface defined by National Semiconductor[2]. The protocol content is essentially a subset of SPI (CPOL = 0, CPHA = 0). Take the Atmel AT93C46D EEPROM as an example and use the Chip Select (CS) pin as the enable. The access is through the 3-wire communication interface, which includes Data Input (DI), Data Output (DO) and Shift Clock (SK).

1.4.1 Atmel AT93C46D EEPROM Communication Protocol

AT93C46D EEPROM master-slave architecture, Master can operate this EEPROM through 7 kinds of instructions. The valid instruction first must be the rising edge of the CS pin, then the DI pin must be the Start Bit (Logic 1), followed by the Opcode and memory location and data, as shown in Table 1-1.

Instruction	Start Bit	Opcode	Address	Data	Comments
READ	1	10	A6 – A0	D7 – D0	Reads data stored in memory at specified address.
EWEN	1	00	11XXXXXXX		Write Enable must precede all programming modes.
ERASE	1	11	A6 – A0		Erases memory location.
WRITE	1	01	A6 – A0	D7 – D0	Writes memory location.
ERAL	1	00	10XXXXXXX		Erases all memory locations.
WRAL	1	00	01XXXXXXX	D7 – D0	Writes all memory locations.
EWDS	1	00	00XXXXXXX		Disables all programming instructions.

Table 1-1 AT93C46D Instruction Set

1.4.2 Basic Timing

The timings of Hold Time and Setup Time between each pin are based on the working environment conditions of 1.8 V ~ 5.5 V. The specifications are shown in Table 1-2 and Figure 1-5. For more detailed specifications, please refer to the specification of AT93C46D[1].

Symbol	Parameter	Min	Max	Units
f_{SK}	SK Clock Frequency		250	kHz
t_{CSS}	CS Setup Time	200		ns
t_{CSH}	CS Hold Time	0		ns
t_{DIS}	DI Setup Time	400		ns
t_{DIH}	DI Hold Time	400		ns
t_{PD0}	Output Delay to 0	1000		ns
t_{PD1}	Output Delay to 1	1000		ns
t_{SV}	CS to Status Valid		1000	ns
t_{DF}	CS to DO in High-impedance		400	ns

Table 1-2 AT93C46D Characteristics

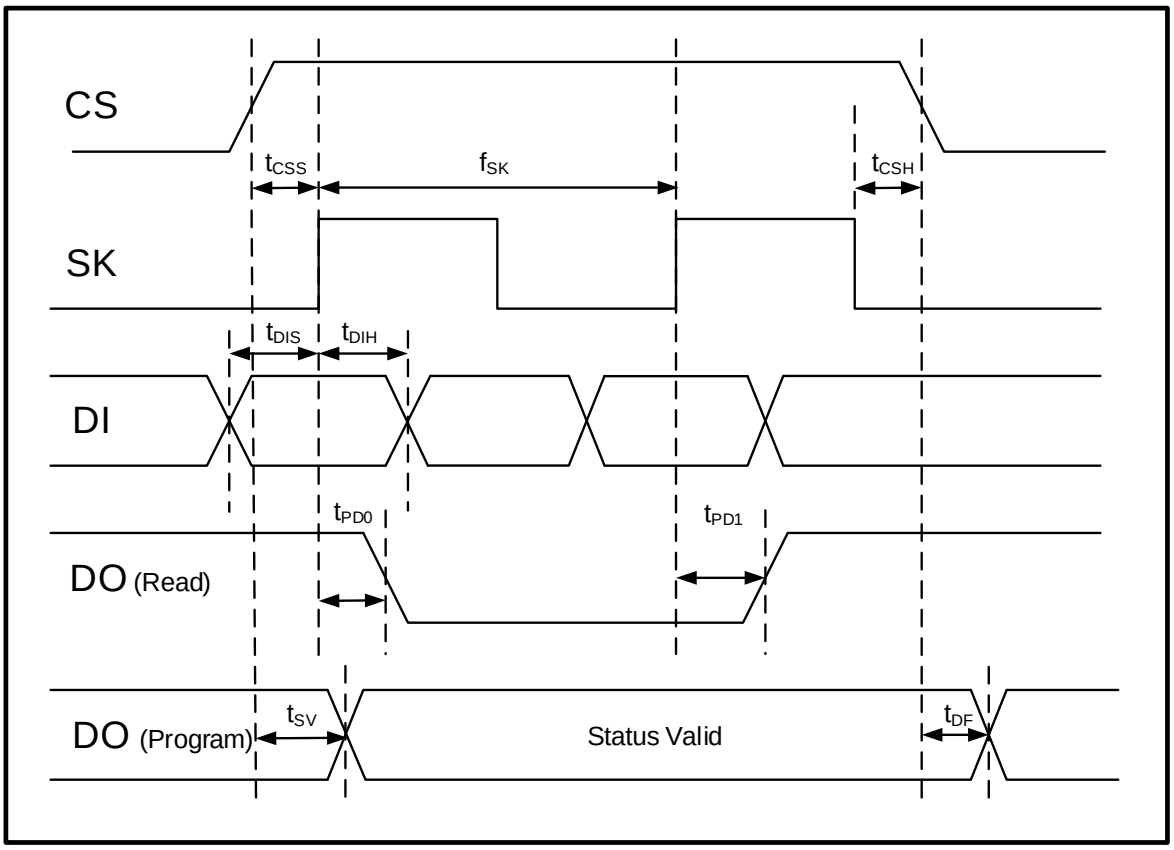

Figure 1-5 Synchronous Data Timing

According to the above 7 kinds of operation instruction rules, this article takes the two sequences of writing and reading as an example to illustrate the timing relationship between each pin.

- **Write Timing**

When the operation command is WRITE, the PSIO first outputs the CS pin high, and then SK starts to output the clock signal. The DI pin is the output bit (Start bit) + 01 (Opcode) + A6 ~ A0 (Address) + D7 ~ D0. After the write operation is completed, the CS pin can be output to high, and then the DO pin status is confirmed as Ready, as shown in Figure 1-6.

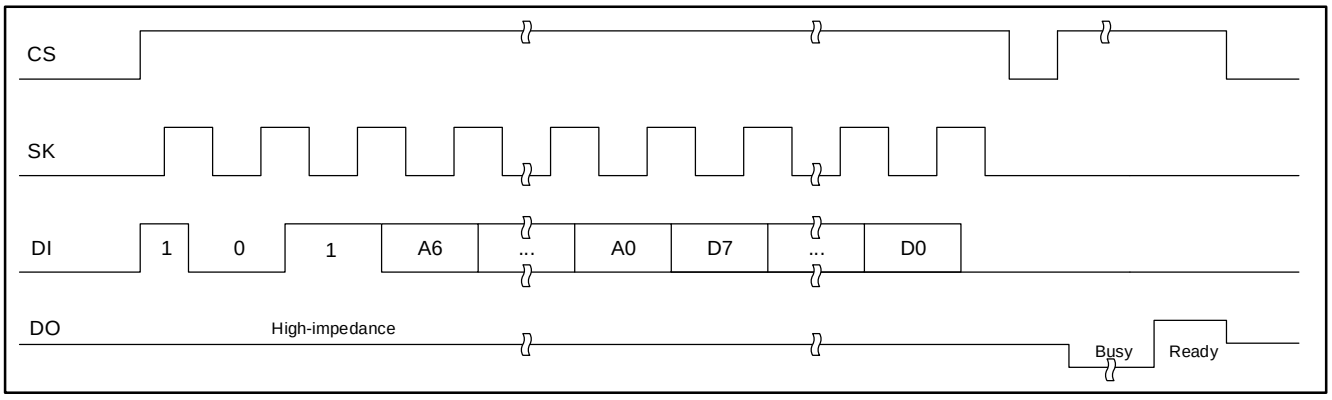

Figure 1-6 Write Timing

- **Read Timing**

When the operation command is READ, the PSIO first outputs the CS pin to high, and then the SK starts to output the clock signal. The DI pin is output 1 (Start bit) + 10 (Opcode) + A6 ~ A0 (Address), and the DO pin is connected to D7 ~ D0 (Data), as shown in Figure 1-7.

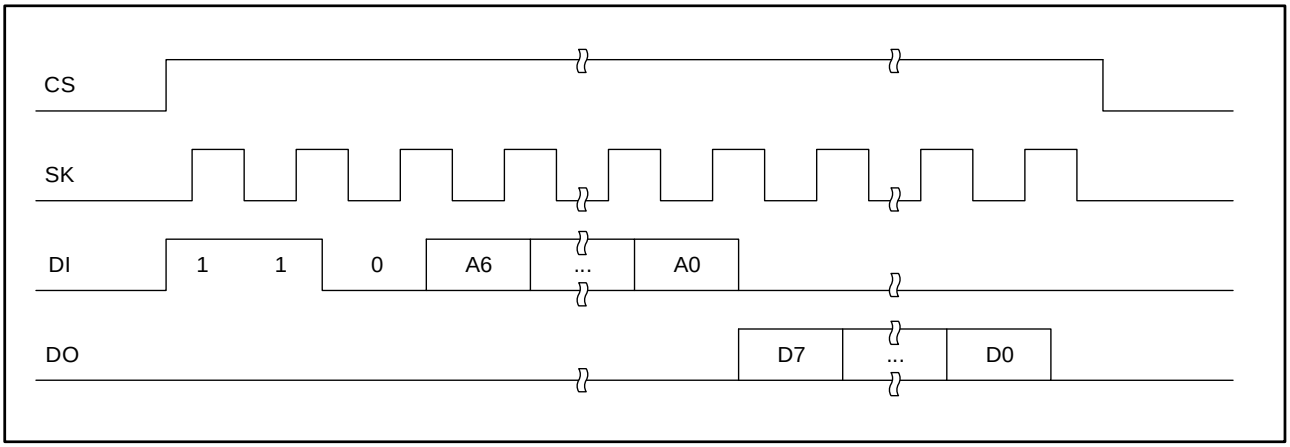

Figure 1-7 Read Timing

1.5 Implementing the Microwire Protocol Using PSIO

The PSIO provides slot controller control timing (slot) time, operation mode, trigger mode, etc. You can adjust the Pin initial state, the Pin state of the slot interval, and the Pin final state by setting the Pin state. The PSIO also provides 4 interrupt functions and support for PDMA functions.

Using PSIO to implement the Microwire protocol can be divided into the following two steps:

1. Find out the basic timing rules for the Microwire agreement. Because of the PSIO operation, you must first set the timing rules (slot) and operate according to the expected timing.
2. Set the operation of Pin. The PSIO can set the check point on the planned timing rule (slot). The check point can set Pin action as output, input and so on.

The PSIO has 4 slot controllers. Each slot controller provides 8 slots, and each slot can count 0 to 15 PSIO engine clocks. In an example of Figure 1-8, using three slot (slot 0, slot 1 and slot 2), slot timing are set to slot 0 = 2 (engine clock), slot 1 = 1 (engine clock), slot 2 = 1 (engine clock). Check point corresponds to 3 slots, check point 0 corresponds to slot 0, check point 1 corresponds to slot 1, check point 2 corresponds to slot 2, and check point 0 sets Pin action to output low level, check point 1 sets Pin action to output high level, check point 2 set Pin action to output low level.

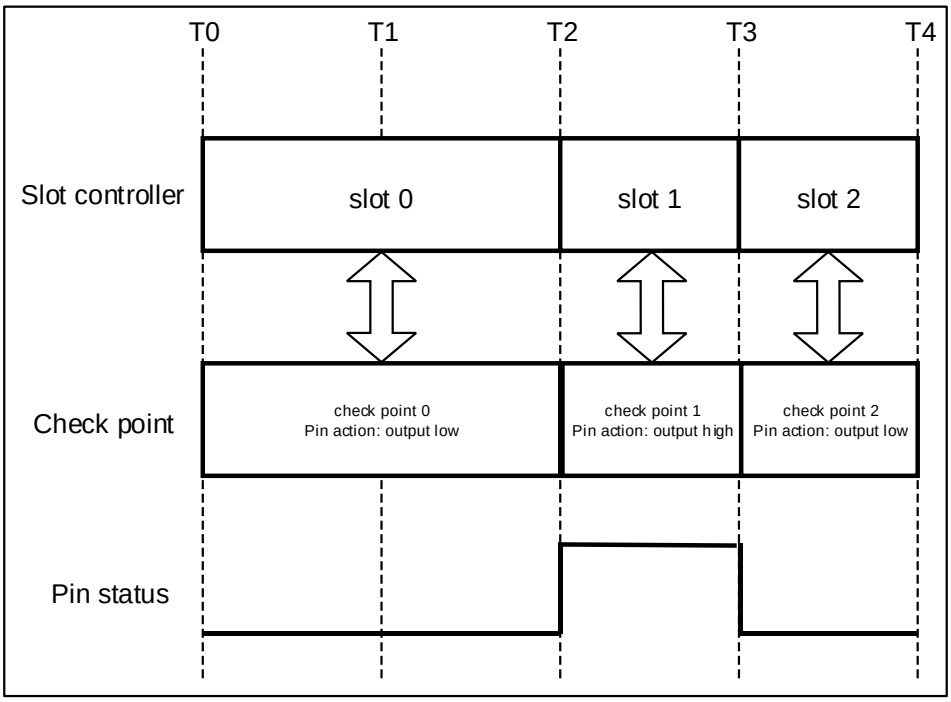

Figure 1-8 PSIO Basic Setting

1.5.1 Establishing Basic Time Unit of the Microwire Protocol

According to 1.4.2, the clock frequency is 250 kHz, and dividing the time unit into 2 μs can produce the required waveform, but in order to preserve the flexibility of the fine-tuning time, this example plans the basic time unit to be 250 ns.

250 ns = PSIO engine clock (HIRC = 48 MHz) divided by 12

- **main.c – SYS_Init()**

```
/* Select PSIO module clock source as HIRC and PSIO module clock divider as 12 */
CLK_SetModuleClock(PSIO_MODULE, CLK_CLKSEL2_PSIOSEL_HIRC, CLK_CLKDIV1_PSIO(12));
```

1.5.2 Planning Output Timing

The Atmel AT93C46D specification the CS pin must first output high, and the timing between the first rising edge of the SK pin is at least 200 ns. Here, 10 engine clocks of 2500 ns are planned from T0 to T9. The frequency of the SK pin is 250 kHz, which is 4 μs, and the T9 ~ T25 has 16 engine clocks, that is 4 μs. After the end of output signal, the CS pin will output high again. The AT93C46D will respond to the completion of the write through the DO pin. Here, using the GPIO function to complete the monitoring of the change of the DO pin makes the program easier, as shown in Figure 1-9.

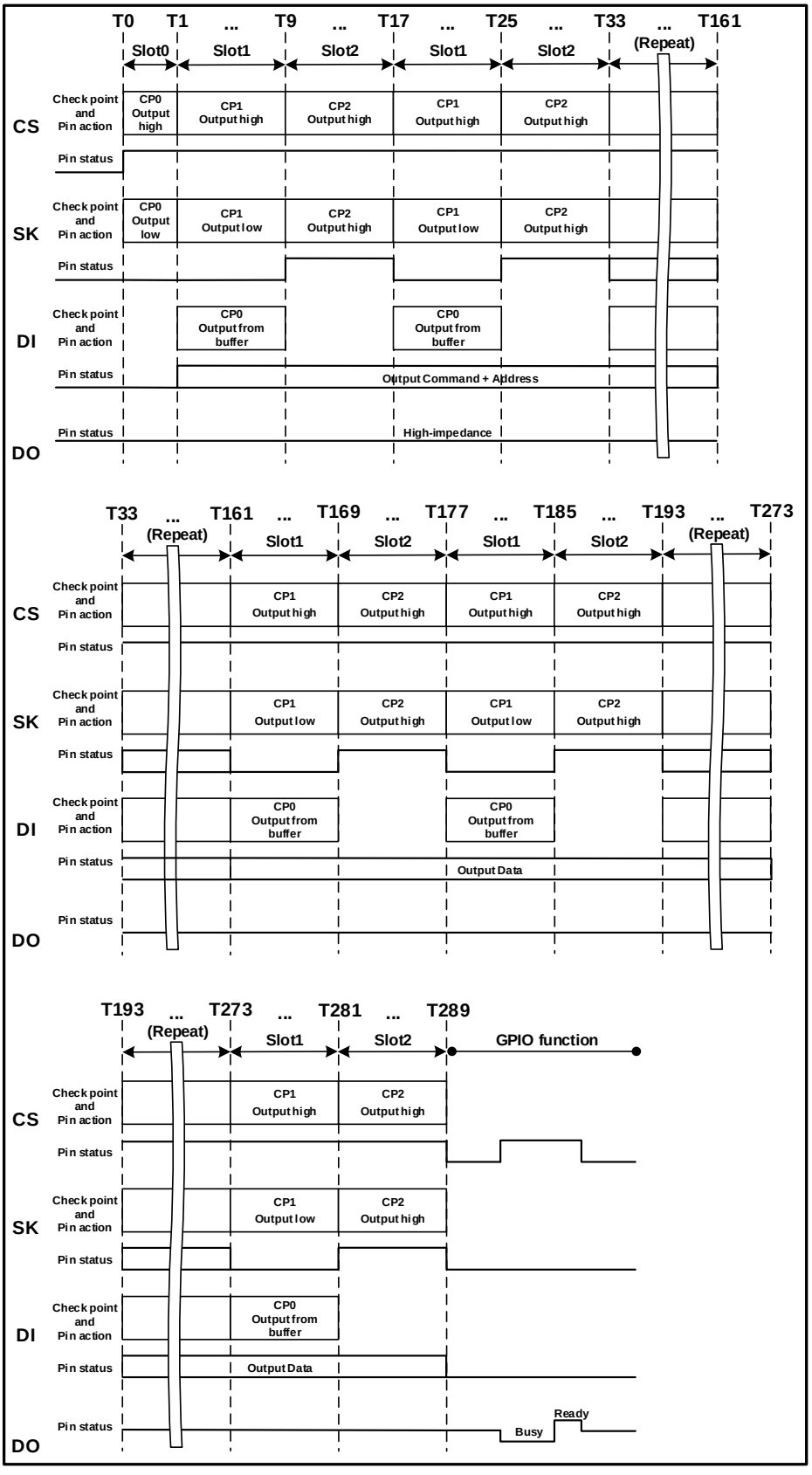

Figure 1-9 Write Timing with PSIO

1.5.3 Planning Input Timing

The input timing of the Atmel AT93C46D is similar to the output timing. The timing planning of pins such as CS, SK, and DI is the same as that of 1.5.2. The difference is that the DO pin must output Command and Address in the DI pin, that is, start reading 8-bits of data after T161. The delay time of the DO pin of the Atmel AT93C46D specification is 1000 ns after the rising edge of the SK pin. Take T169 ~ T177 as an example, there are 8 engine clocks. According to the PSIO specification[3], the data will be captured at the 6th engine clock, that is, 6 engine clock * 250 ns is 1500 ns, which conforms to the specifications of Atmel AT93C46D, as shown in Figure 1-10.

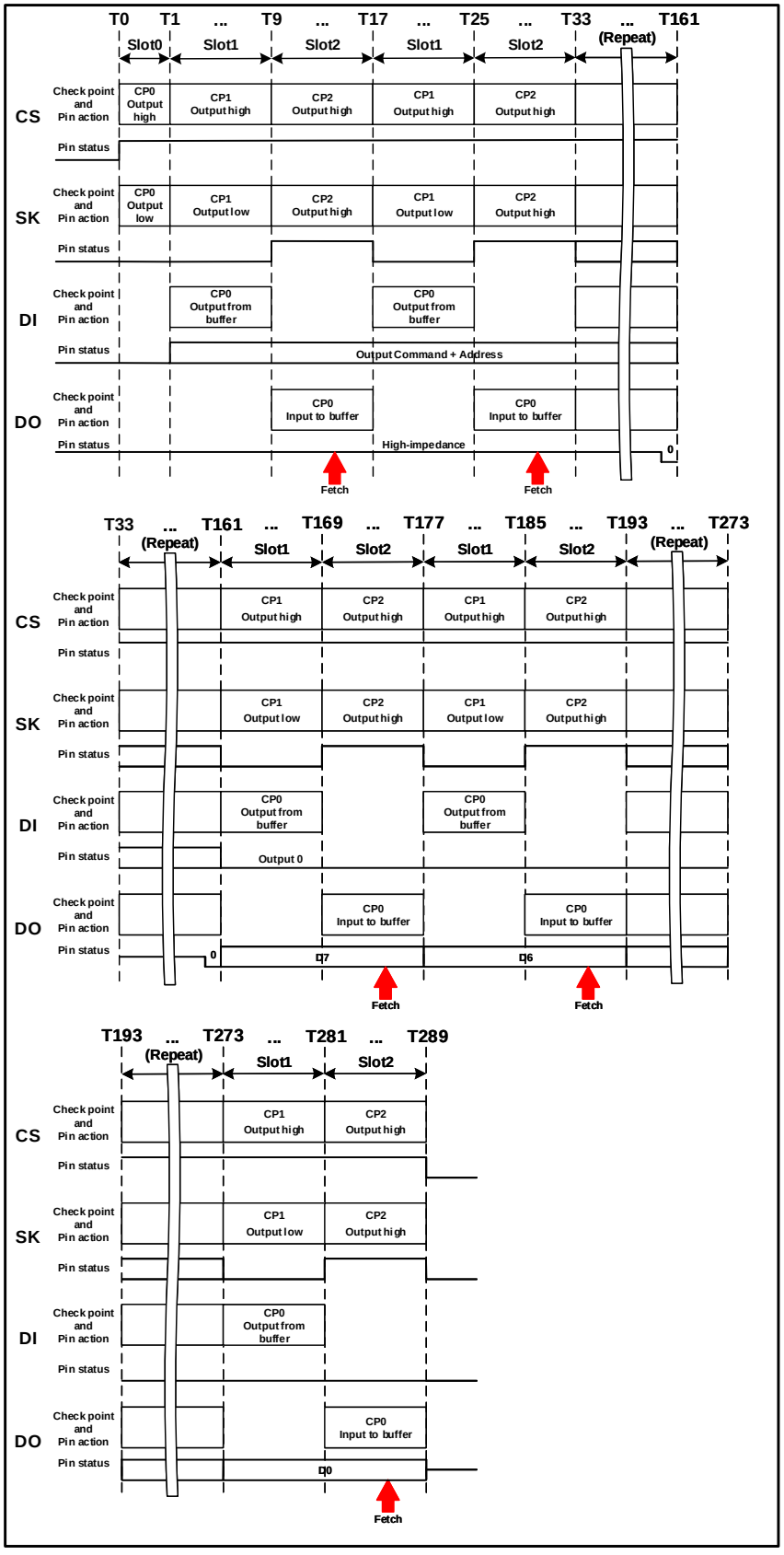

Figure 1-10 Read Timing with PSIO

2 Microwire Sample Code

This sample program demonstrates how to implement Microwire communication protocol through PSIO. The main.c is the program control flow, including device initialization, erasing, reading and writing EEPROM content, as shown in Figure 2-1. AT93C46D EEPROM detailed specifications can refer to the specification[1].

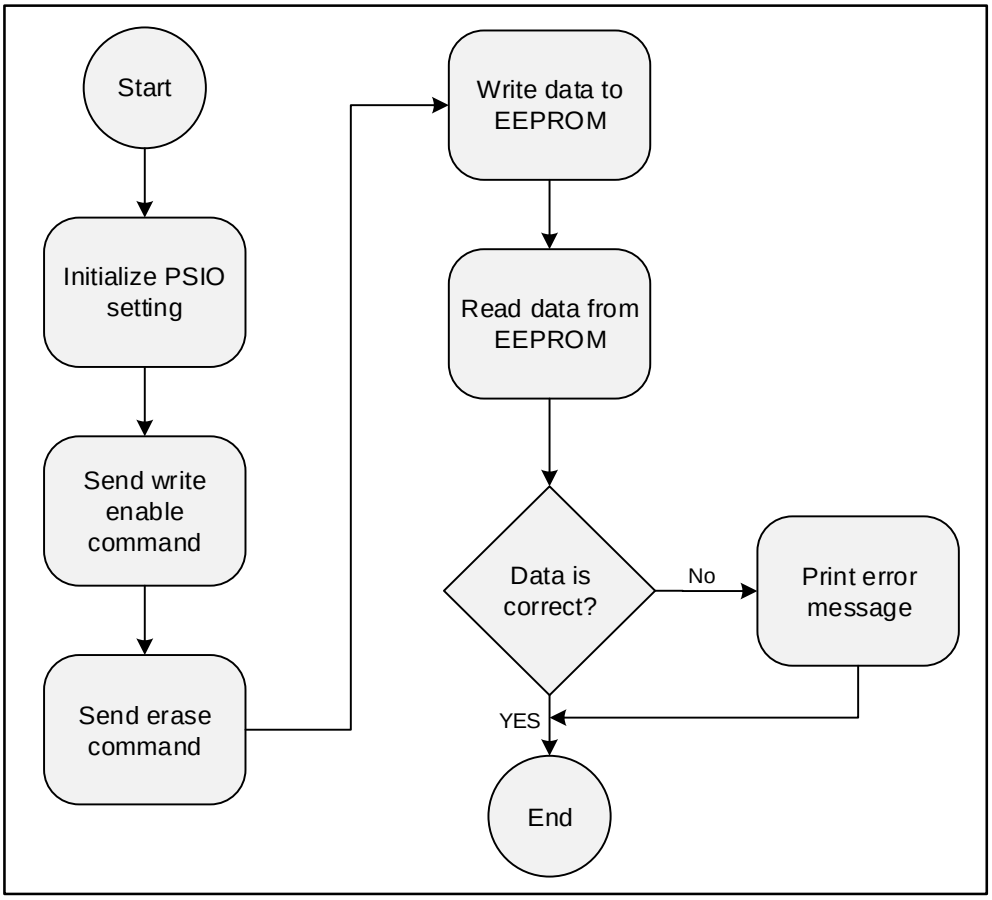

Figure 2-1 Microwire Sample Code Flowchart

- **main.c – main( ): Initialize and write enable command**

```
int main(void)
{
    . . . . .
    /* Use slot controller 0, pin 0, pin1, pin 2, pin 3 */
    sConfig.u8SlotCtrl              = PSIO_SC0;
    sConfig.u8ChipSelectPin        = PSIO_PIN0;
    sConfig.u8ClockPin              = PSIO_PIN1;
    sConfig.u8DO                    = PSIO_PIN2;
    sConfig.u8DI                    = PSIO_PIN3;
```

```
/* Initialize PIN config */
pu32ChipSelectPin     = &PB15;
pu32InputPin          = &PC3;

/* Set CS pin output mode when GPIO function */
GPIO_SetMode(PB, BIT15, GPIO_MODE_OUTPUT);

/* Initialize PSIO setting for AT93C46D */
PSIO_AT93C46D_Init(&sConfig);

/* Send write enable command */
PSIO_AT93C46D_EraseWrite_Enable(&sConfig);
. . . . .
```

- **main.c – main( ): Erase/Read/Write EEPROM**

```
. . . . .
printf("Read/Write data to AT93C46D\n");

for (u8Address=0; u8Address<EEPROM_SIZE/DATA_WIDTH; u8Address++,u8TxData =u8Address+2)
{
    /* Send erase command */
    PSIO_AT93C46D_Erase(&sConfig, u8Address);

    /* Write data to AT93C46D */
    PSIO_AT93C46D_Write(&sConfig, u8Address, &u8TxData);

    /* Read data from AT93C46D */
    PSIO_AT93C46D_Read(&sConfig, u8Address, &u8RxData);

    /* Compare data is correct */
    if (u8TxData != u8RxData)
    {
        printf("[Error] TxData:0x%x, RxData:0x%x\n", u8TxData, u8RxData);
        while (1);
    }
    else
    {
```

```
            printf(".");
        }
    }
    . . . . .
```

2.1 PSIO Initialization

First, during initialization, use PSIO_SET_GENCTL() to set the state of Pin. The contents are Pin, Slot controller, mode, initialization state and end state. In this example, CS, SK, DI, DO are set in the initialization, and the CS, SK, and DI pins are turned on. The DO pin is turned on when the read program is executed, so as to avoid reading the data continuously while the output is not processed, and the PSIO operation is stopped. PSIO_SCSLOT_SET_SLOT() is used to set the count number of each slot. Set slot0 to 1, slot2 to 8, and slot3 to 8. PSIO_SET_CHECKPOINT() sets the correspondence between the slot and the check point, and sets the pin position action of the check point through PSIO_SET_ACTION(). The settings of CS, SK, DI, and DO in this example are shown in Table 2-1.

Pin	Slot 0	Slot 1	Slot 2
CS	**[Check Point]** 0 **[Action]** Output high	**[Check Point]** 1 **[Action]** Output high	**[Check Point]** 2 **[Action]** Output high
SK	**[Check Point]** 0 **[Action]** Output low	**[Check Point]** 1 **[Action]** Output low	**[Check Point]** 2 **[Action]** Output high
DI	Disable	**[Check Point]** 0 **[Action]** Output buffer	Disable
DO	Disable	Disable	**[Check Point]** 0 **[Action]** Input buffer

Table 2-1 PSIO Initialization Settings

- **AT93C46D_driver_EEPROM.c – PSIO_AT93C46D_Init ( )**

```
void PSIO_AT93C46D_Init(S_PSIO_AT93C46D *psConfig)

{

    const S_PSIO_CP_CONFIG sCSConfig

    ={/* Check Point0   Check Point1   Check Point2   Check Point3       Check Point4       Check Point5       Check Point6       Check Point7 */

/* Slot */ PSIO_SLOT0,    PSIO_SLOT1,    PSIO_SLOT2,    PSIO_SLOT_DISABLE, PSIO_SLOT_DISABLE, PSIO_SLOT_DISABLE, PSIO_SLOT_DISABLE, PSIO_SLOT_DISABLE,

/* Action */ PSIO_OUT_HIGH, PSIO_OUT_HIGH, PSIO_OUT_HIGH, PSIO_NO_ACTION,    PSIO_NO_ACTION,    PSIO_NO_ACTION,    PSIO_NO_ACTION,    PSIO_NO_ACTION};
```

```
    const S_PSIO_CP_CONFIG sSKConfig
    = {/* Check Point0  Check Point1  Check Point2   Check Point3       Check Point4       Check Point5       Check Point6       Check Point7 */
/* Slot */ PSIO_SLOT0,   PSIO_SLOT1,   PSIO_SLOT2,    PSIO_SLOT_DISABLE, PSIO_SLOT_DISABLE, PSIO_SLOT_DISABLE, PSIO_SLOT_DISABLE, PSIO_SLOT_DISABLE,
/* Action */ PSIO_OUT_LOW, PSIO_OUT_LOW, PSIO_OUT_HIGH, PSIO_NO_ACTION,    PSIO_NO_ACTION,    PSIO_NO_ACTION,    PSIO_NO_ACTION,    PSIO_NO_ACTION};
    const S_PSIO_CP_CONFIG sDIConfig
    = {/* Check Point0    Check Point1      Check Point2      Check Point3      Check Point4      Check Point5      Check Point6      Check Point7 */
/* Slot */PSIO_SLOT1,PSIO_SLOT_DISABLE,PSIO_SLOT_DISABLE,PSIO_SLOT_DISABLE, PSIO_SLOT_DISABLE, PSIO_SLOT_DISABLE, PSIO_SLOT_DISABLE, PSIO_SLOT_DISABLE,
/* Action */PSIO_OUT_BUFFER, PSIO_NO_ACTION, PSIO_NO_ACTION, PSIO_NO_ACTION,PSIO_NO_ACTION,    PSIO_NO_ACTION,    PSIO_NO_ACTION,    PSIO_NO_ACTION};
    const S_PSIO_CP_CONFIG sDOConfig
    = {/* Check Point0    Check Point1      Check Point2      Check Point3      Check Point4      Check Point5       Check Point6       Check Point7 */
/* Slot */PSIO_SLOT2,PSIO_SLOT_DISABLE,PSIO_SLOT_DISABLE,PSIO_SLOT_DISABLE, PSIO_SLOT_DISABLE, PSIO_SLOT_DISABLE, PSIO_SLOT_DISABLE, PSIO_SLOT_DISABLE,
/* Action */PSIO_IN_BUFFER, PSIO_NO_ACTION, PSIO_NO_ACTION, PSIO_NO_ACTION, PSIO_NO_ACTION,    PSIO_NO_ACTION,    PSIO_NO_ACTION,    PSIO_NO_ACTION};
   . . . . .
```

```
void PSIO_AT93C46D_Init(S_PSIO_AT93C46D *psConfig)
{
    . . . . .
    /* Enable CS, SK, DI, DO pin and setting general configuration */
    PSIO_SET_GENCTL(PSIO, psConfig->u8ChipSelectPin, PSIO_PIN_ENABLE, psConfig->u8SlotCtrl
                    , PSIO_OUTPUT_MODE, PSIO_LOW_LEVEL, PSIO_LOW_LEVEL);
    PSIO_SET_GENCTL(PSIO, psConfig->u8ClockPin, PSIO_PIN_ENABLE, psConfig->u8SlotCtrl
                    , PSIO_OUTPUT_MODE, PSIO_LOW_LEVEL, PSIO_LOW_LEVEL);
    PSIO_SET_GENCTL(PSIO, psConfig->u8DI, PSIO_PIN_ENABLE, psConfig->u8SlotCtrl
                    , PSIO_OUTPUT_MODE, PSIO_LOW_LEVEL, PSIO_LOW_LEVEL);
    PSIO_SET_GENCTL(PSIO, psConfig->u8DO, PSIO_PIN_DISABLE, psConfig->u8SlotCtrl
                    , PSIO_INPUT_MODE, PSIO_LOW_LEVEL, PSIO_LOW_LEVEL);

    /* Set data order and MSB */
    PSIO_SET_ORDER(PSIO, psConfig->u8ChipSelectPin, PSIO_MSB);
    PSIO_SET_ORDER(PSIO, psConfig->u8ClockPin, PSIO_MSB);
    PSIO_SET_ORDER(PSIO, psConfig->u8DO, PSIO_MSB);
    PSIO_SET_ORDER(PSIO, psConfig->u8DI, PSIO_MSB);

    /* Set slot counter number */
    PSIO_SCSLOT_SET_SLOT(PSIO, psConfig->u8SlotCtrl, PSIO_SLOT0, 1);
```

```
    PSIO_SCSLOT_SET_SLOT(PSIO, psConfig->u8SlotCtrl, PSIO_SLOT1, 8);
    PSIO_SCSLOT_SET_SLOT(PSIO, psConfig->u8SlotCtrl, PSIO_SLOT2, 8);

    /* Set check point configuration */
    PSIO_SET_CP_CONFIG(PSIO, psConfig->u8ChipSelectPin, &sCSConfig);
    PSIO_SET_CP_CONFIG(PSIO, psConfig->u8ClockPin, &sSKConfig);
    PSIO_SET_CP_CONFIG(PSIO, psConfig->u8DI, &sDIConfig);
    PSIO_SET_CP_CONFIG(PSIO, psConfig->u8DO, &sDOConfig);
}
```

2.2 Output Data

This article has three examples of output data, namely:

1. PSIO_AT93C46D_EraseWrite_Enable()
2. PSIO_AT93C46D_Erase ()
3. PSIO_AT93C46D_Write()

The timing setting of the data output is the same, and the difference is the number of bits of the output data, PSIO_AT93C46D_EraseWrite_Enable() output 12-bit、PSIO_AT93C46D_Erase () output 10-bit、PSIO_AT93C46D_Write() output 18-bit. Using PSIO implementation only needs to adjust the number of repetitions of slot1 and slot2, and the width of the data buffer register. In addition, PSIO output data uses CS, SK, DI and other pins, and the confirmation status PSIO_AT93C46D_CheckStatus() is implemented using GPIO function.

- **AT93C46D_driver_EEPROM.c – PSIO_AT93C46D_ EraseWrite_Enable ( )**

```
void PSIO_AT93C46D_EraseWrite_Enable(S_PSIO_AT93C46D *psConfig)
{
    /* Loop slot1~slot2 11 times */
    PSIO_SET_SCCTL(PSIO, psConfig->u8SlotCtrl, PSIO_SLOT1
                   , PSIO_SLOT2, 11, PSIO_REPEAT_DISABLE);

    /* Set output data width as 12 */
    PSIO_SET_WIDTH(PSIO, psConfig->u8DI, 0, 12);

    /* Set output data */
    PSIO_SET_OUTPUT_DATA(PSIO, psConfig->u8DI, CMD_EWEN);

    /* Trigger slot controller */
    PSIO_START_SC(PSIO, psConfig->u8SlotCtrl);

    /* Wait for slot controller is not busy */
    while (PSIO_GET_BUSY_FLAG(PSIO, psConfig->u8SlotCtrl));
}
```

- **AT93C46D_driver_EEPROM.c – PSIO_AT93C46D_ Write ( )**

```
void PSIO_AT93C46D_Erase(S_PSIO_AT93C46D *psConfig, uint8_t u8Address)
{
    /* Loop slot1~slot2 9 times */
    PSIO_SET_SCCTL(PSIO,psConfig->u8SlotCtrl,PSIO_SLOT1,PSIO_SLOT2,9,PSIO_REPEAT_DISABLE);
```

```
    /* Set output data width as 10 */
    PSIO_SET_WIDTH(PSIO, psConfig->u8DI, 0, 10);

    /* Set output data */
    PSIO_SET_OUTPUT_DATA(PSIO, psConfig->u8DI, CMD_ERASE | (u8Address & 0x7F));

    /* Trigger slot controller */
    PSIO_START_SC(PSIO, psConfig->u8SlotCtrl);

    /* Wait for slot controller is not busy */
    while (PSIO_GET_BUSY_FLAG(PSIO, psConfig->u8SlotCtrl));

    /* Check device status is READY */
    PSIO_AT93C46D_CheckStatus(psConfig);
}
```

- **AT93C46D_driver_EEPROM.c – PSIO_AT93C46D_ Erase ( )**

```
void PSIO_AT93C46D_Erase(S_PSIO_AT93C46D *psConfig, uint8_t u8Address)
{
    /* Loop slot1~slot2 9 times */
    PSIO_SET_SCCTL(PSIO,psConfig->u8SlotCtrl,PSIO_SLOT1,PSIO_SLOT2,9,PSIO_REPEAT_DISABLE);

    /* Set output data width as 10 */
    PSIO_SET_WIDTH(PSIO, psConfig->u8DI, 0, 10);

    /* Set output data */
    PSIO_SET_OUTPUT_DATA(PSIO, psConfig->u8DI, CMD_ERASE | (u8Address & 0x7F));

    /* Trigger slot controller */
    PSIO_START_SC(PSIO, psConfig->u8SlotCtrl);

    /* Wait for slot controller is not busy */
    while (PSIO_GET_BUSY_FLAG(PSIO, psConfig->u8SlotCtrl));

    /* Check device status is READY */
    PSIO_AT93C46D_CheckStatus(psConfig);
}
```

- **AT93C46D_driver_EEPROM.c – PSIO_AT93C46D_ CheckStatus ( )**

```
void PSIO_AT93C46D_CheckStatus(S_PSIO_AT93C46D *psConfig)
{
    /* Set chip select pin to GPIO function */
    SetCSPinToGPIO();

    /* Check device status is READY */
    outpw(pu32ChipSelectPin, 1);
    while(inpw(pu32InputPin) == 1);
    while(inpw(pu32InputPin) == 0);
    outpw(pu32ChipSelectPin, 0);

    /* Set chip select pin to PSIO function */
    SetCSPinToPSIO();
}
```

2.3 Input Data

The implementation of the input data needs to first output 10-bit command + address, followed by 8-bit data. In order to keep the timing uninterrupted, after the output 10-bit is completed, the PSIO is not reset and restarted, but is regarded as a complete 18-bit action, so after the 10-bit output is completed, continue to output Low. Data is to discard the 10-bit received at the beginning, leaving only bit 10 ~ bit 17 as valid data, as shown in Figure 2-2.

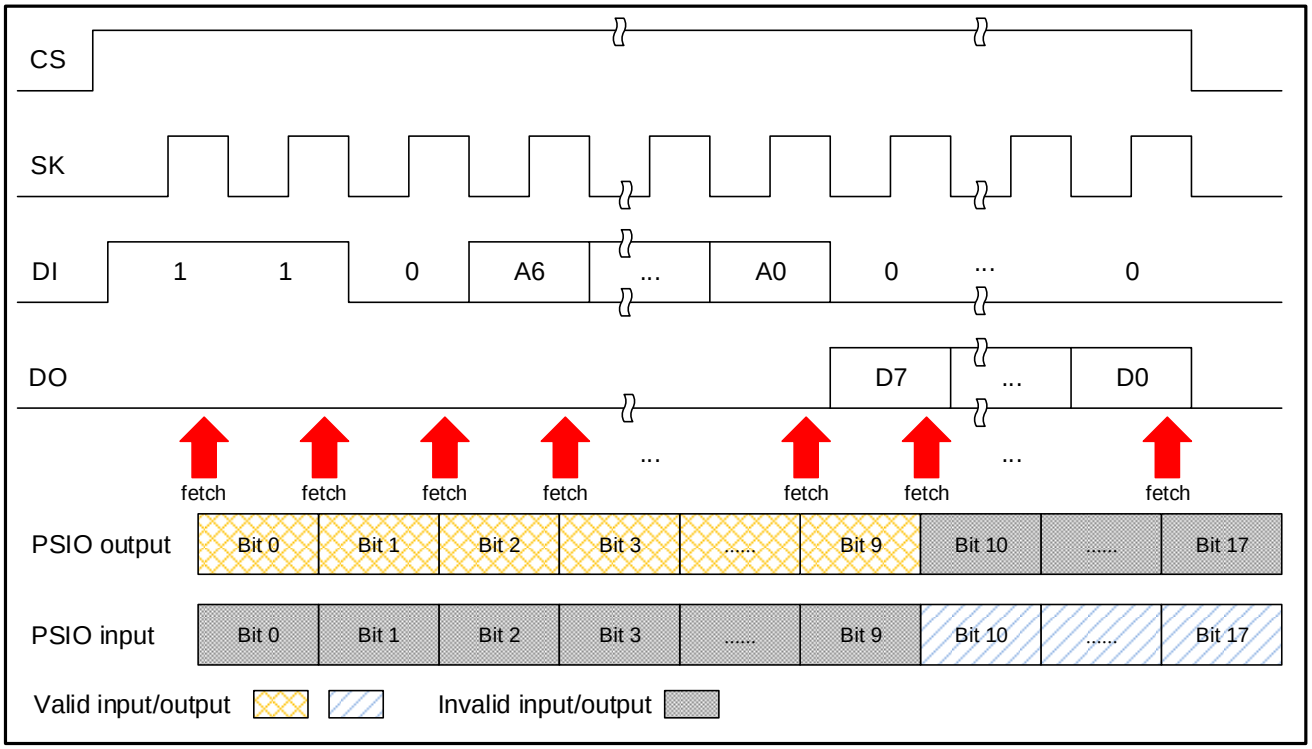

Figure 2-2 Implementing Read Function with PSIO

In order to avoid malfunction, as explained in section 2.1, the DO pin is turned on at the input, and turned off at the end of the input. A total of 18 bits require input and output, so the number of repetitions of slot1 and slot2 is set to 17 (18 times in total), and the width of the data buffer register is set to 18.

- **AT93C46D_driver_EEPROM.c – PSIO_AT93C46D_ Read ( )**

```
void PSIO_AT93C46D_Read(S_PSIO_AT93C46D *psConfig, uint8_t u8Address, uint8_t *pu8Data)
{
    uint32_t u32Data;

    /* Enable DO pin */
    PSIO_ENABLE_PIN(PSIO, psConfig->u8DO);
```

```
    /* Loop slot1~slot2 17 times */
    PSIO_SET_SCCTL(PSIO, psConfig->u8SlotCtrl, PSIO_SLOT1, PSIO_SLOT2, 17
                   , PSIO_REPEAT_DISABLE);
    /* Set output/input data width as 18 */
    PSIO_SET_WIDTH(PSIO, psConfig->u8DI, 0, 18);
    PSIO_SET_WIDTH(PSIO, psConfig->u8DO, 18, 0);

    /* Set output data */
    PSIO_SET_OUTPUT_DATA(PSIO, psConfig->u8DI, CMD_READ(u8Address));

    /* Trigger slot controller */
    PSIO_START_SC(PSIO, psConfig->u8SlotCtrl);

    /* Wait for data buffer is full */
    while (!PSIO_GET_TRANSFER_STATUS(PSIO, PSIO_TRANSTS_INFULL0_Msk<<(psConfig->u8DO*4)));

    /* Read data from device */
    u32Data = PSIO_GET_INPUT_DATA(PSIO, psConfig->u8DO);
    *pu8Data = u32Data & 0xFF;

    /* Disable DO pin */
    PSIO_DISABLE_PIN(PSIO, psConfig->u8DO);
}
```

3 Sample Program Verification

This sample program runs on the Nuvoton M251/M252 series microcontrollers and is connected to the Atmel AT93C46D EEPROM. In addition to verifying that the Atmel AT93C46D EEPROM can be successfully operated in this section, and using the Logic Analyzer[4], the signal analysis is performed during the transmission of the signal, as shown in Figure 3-1.

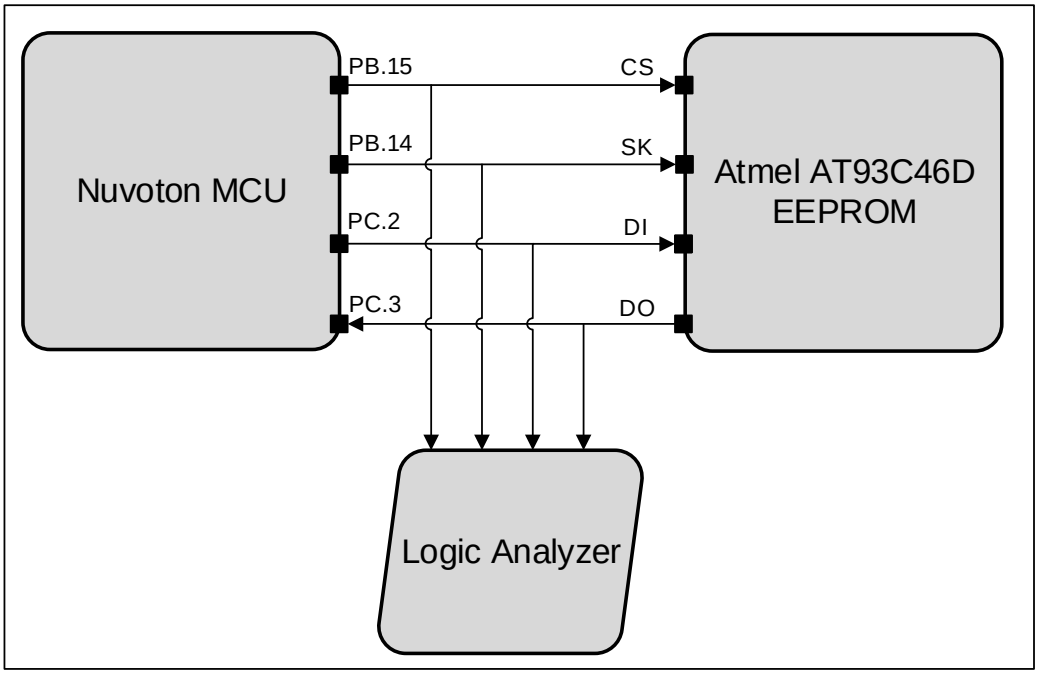

Figure 3-1 System Block Diagram

3.1 Verify Output Waveform

Main.c – main( )

```
. . . . .
printf("Read/Write data to AT93C46D\n");


for (u8Address=0; u8Address<EEPROM_SIZE/DATA_WIDTH; u8Address++,u8TxData =u8Address+2)
{
    . . . . .
    /* Write data to AT93C46D */
    PSIO_AT93C46D_Write(&sConfig, u8Address, &u8TxData);
    . . . . .
}
. . . . .
```

Taking this code as an example, to write data to the specified address location, the waveform of the DI pin is Command + Address(7-bit) + Data(8-bit), where Command (Start bit + Opcode) is 3'b101, which is the Write field in Figure 3-2. When u8Address is 0xA, write Data to 0xC, and use the logic analyzer to measure the result as shown in Figure 3-2.

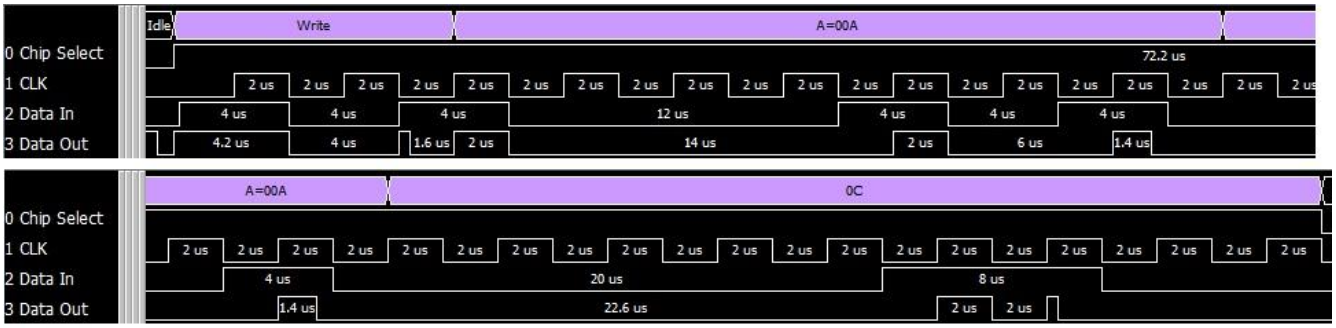

Figure 3-2 The Timing of Write One Byte

3.2 Verify Input Waveform

Main.c – main( )

```
. . . . .
printf("Read/Write data to AT93C46D\n");


for (u8Address = 0; u8Address < EEPROM_SIZE/DATA_WIDTH; u8Address++, u8TxData++)
{
    . . . . .
    /* Read data from AT93C46D */
    PSIO_AT93C46D_Read(&sConfig, u8Address, &u8RxData);
    . . . . .
}
. . . . .
```

Taking this code as an example, to read data to the specified address location, the waveform of the DI pin is Command + Address (7-bit), and then the waveform of the DO pin is Data (8-bit). When u8Address is 0xA, the result measured by the logic analyzer is shown in Figure 3-3, the DI pin is Command (Start bit + Opcode) is 3'b110, Address is 0xA, and then the DO pin Data is 0xC.

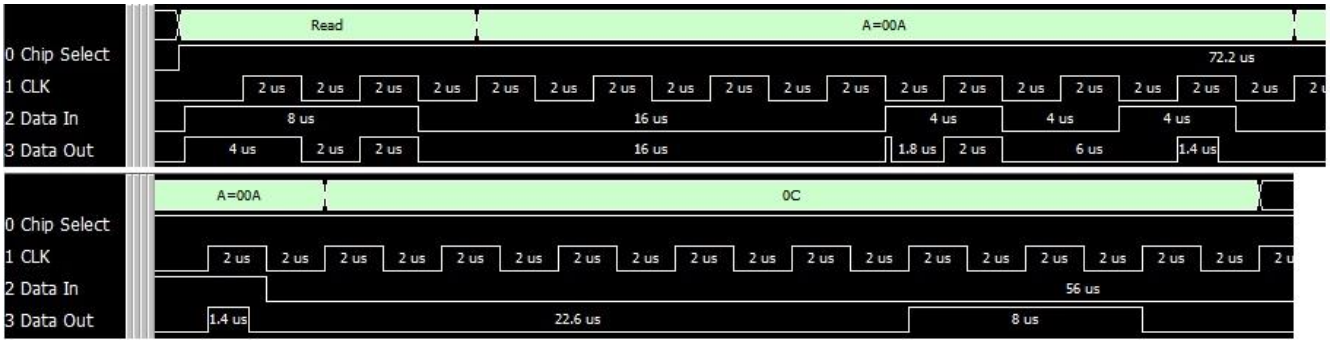

Figure 3-3 The Timing of Read One Byte

4 Conclusion

This article uses the M251/M252 PSIO to implement the Microwire protocol and connect to the AT93C46D device for operation. Through the read and write memory verification, the instrument measurement method proves that the Microwire protocol implemented by M251/ M252 PSIO can successfully communicate with the AT93C46D device.

Using PSIO to implement the Microwire protocol is not the only way. For example, GPIO, Timer and software control can achieve the same purpose. However, Microwire has 4 pins. If you need to toggle the pin through GPIO at the same time, the program will become complicated. With the PSIO implementation, controlling the pin change can be done with the appropriate PSIO settings.

5 Reference

[1] Atmel AT93C46D EEPROM, http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-5193-SEEPROM-AT93C46D-Datasheet.pdf

[2] Microwire, http://www.ti.com/lit/an/snoa743/snoa743.pdf

[3] Nuvoton M251/M252 Technical Reference Manual, PSIO, PSIOn Input Data Register (PSIOn_INDAT)

[4] Acute TL2236B, http://www.acute.com.tw/eng/index.php

Revision History

Date	Revision	Description
2019.05.15	1.00	1. Initially issued.

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.