Introduction

This technical note provides detailed information regarding the ATMEL AT25080A / 160A / 320A / 640A EEPROMs installation including a description of the hardware connections and data loading process from an external EEPROM.

The use of the external EEPROM is optional. Typically, an EEPROM code is used to upgrade system capabilities, change register default values, or change pre-configuration registers.

This technical note relates to systems working in Auto Mode. In systems using the Enhanced Mode application, the Microsemi™ MCU System Manager performs the parameter updates and patch code loading.

Applicable Documents

- PD69012 Data Sheet, Cat. No. 06-0069-058
- Microsemi Application Note AN-174 for Designing a 48-port Enhanced PoE System (802.3af/802.3at Compliant, UART) Cat. No. 06-0054-080
- Microsemi Application Note AN-176, Cat No. 06-0071-080 for Designing a 96-port Auto Mode PoE System
- ATMEL AT25080A/160A/320A/640A data sheet

General Description

Communication with an external EEPROM is based on a standard Serial Peripheral Interface (SPI) bus.

The communication to the external EEPROM is enabled by the Chip Select signal (CS) and accessed via a three-wire interface consisting of Serial Data Input, Serial Data Output, and Serial Clock (MISO, MOSI, SCK).

These lines (wires) are the ESPI interface signals of the system and form a dedicated CS line functioning as the LED Stream Latch signal after the boot sequence.

The lines also function as an SPI communication interface to the EEPROM after power-up during the boot stage of the PD69012. After the boot stage completion, the lines are toggled to function as the main system communication interface. The LSD (LED Stream Data) signal is used to "HOLD" the EEPROM during ongoing operations of the LSL (LED Stream Latch) signal.

The Host can also perform the patch code download and parameters update if the RESET, DISABLE_PORTS and I^2C lines are connected between the Host and the PD69012s.

Hardware Connectivity

Option 1: Stand Alone Auto Mode System with External EEPROM

Upon power up, the Master IC (PD69012) tries to communicate with the external EEPROM. If an EEPROM is connected and contains a valid firmware code, it reads the data and uploads the EEPROM data to all PD69012s (Master and Slaves) connected in the system.

A valid bit in the PD69012 Master is set if a valid EEPROM and data are detected and uploaded successfully. The bit is actually bit [14] in the **SW_BootState** register (address 0x1168).

'1' = valid EEPROM found and data uploaded successfully.

Figure 1 demonstrates the interconnection between the EEPROM and the PD69012 PoE controller.

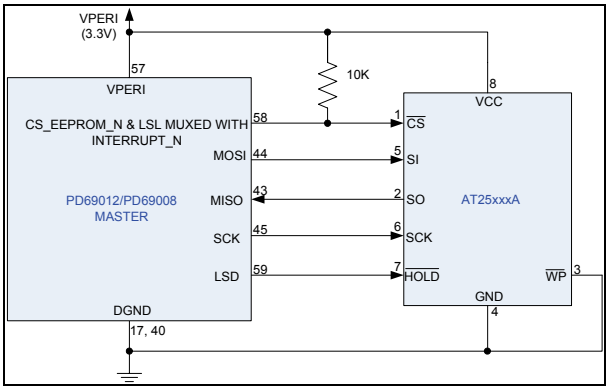

Figure 1: Interconnection between EEPROM and PD69012 PoE Controller

Figure 2 shows the interconnection between the Host and the PD69012 devices.

(Refer to AN176 Cat No. 06-0071-080 for a detailed description of all the PD690012 PoE controller signals reaching the Host CPU).

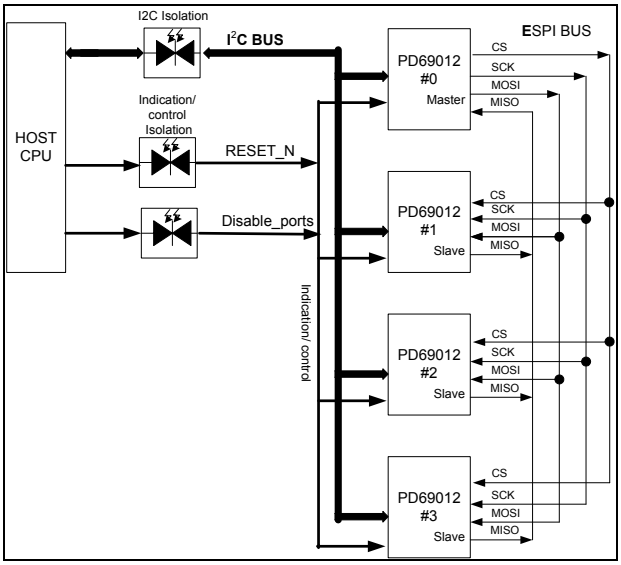

Figure 2: Auto Mode System with Host

Option 2: Auto Mode System with Host

Upon power up, the Host should hold the **disable_ports** line at “low” state (0), while the RESET line is asserted.

This sequence holds the PoE system in the configuration state. In this condition the master IC will stay in **config** mode and will hold all slaves in **config** mode until the Disable ports line is released. During this state all memory map registers are not write-protected and the Host can change them. The Host can also configure the system parameters and download the patch code to the PD69012 devices via the I^2C interface.

Figure 3 describes the method of operation.

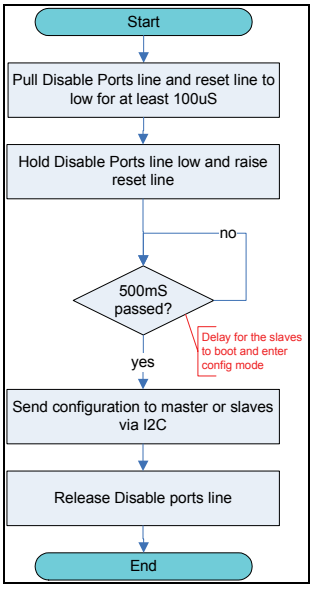

Figure 3: Auto Mode System

Download File (HEX or BIN) Patch Code

The download file is either an HEX file for EEEPROM programming during production, or a BIN file for the Host, which is then uploaded to the PD69012s ICs.

The file contains a header, registers parameters (if required), a patch code and checksum.

Code Example for Host

Following code is an example for the host SW to read the download file and program it to the devices.

```
//constant definitions
#define EEPROM_ID       0x1910
#define BROADCAST_ADD   0x000E
//structure definitions
typedef struct
{
    word    ID;
    word    StructVer;
    word    DataVer;
    word    NumRecords;
    word    EnabledPatches;
    word    DataStartAdd;
    byte    Spare[32];
    word    HeaderCS;
}t_EEpromHeader;

typedef struct
{
    word    ChipNo;
    word    RamAdd;
    word    Data;
}t_EEpromData;
//function prototypes
void EepromRead(word* Dest, word EepromAdd, word NumWords);
void RotemWrite(word ChipAdd, word ChipRamAdd, word* Data, word NumWords);

//=====================================================================
//Function      :   OperateEEprom
//Description   :   this function reads from EEprom, verifies the data and
//                  sends it to slaves
//Parameters    :   none
//Return Value  :   bool - TRUE if successfull
//                         FALSE if error
//Basic assumptions :
//                  1.The EEprom can be substituted by a memory space.
//                  in such case the EepromRead function should be replaced by
//                  a read from the memory space.
//                  2.The rotem chips are numbered in the EEprom data from 0
//                  to 7. master and 7 slaves in A mode or 8 slaves in E mode.
//                  when working via I2C add the base I2C address to the
//                  chip address written in the EEprom.
//=====================================================================
bool OperateEEprom()
{
    t_EEpromHeader Header;
    t_EEpromData Data;
    word    CheckSum = 0;
    word    CurrAdd;
    byte    DataCsIndex;
    word    Index;
    byte*   bPtr;

    //read the header from the EEprom
```

```
    EepromRead((word)&Header, 0x0002, sizeof(t_EEpromHeader) / 2);
    //check the ID
    if(Header.ID != EEPROM_ID)
    {
        //invalid ID
        return FALSE;
    }
    //get the address of the struct object
    bPtr = (byte*)&Header;
    //check the check sum of the header
    for(Index = 0; Index < (sizeof(t_EEpromHeader) - 2); Index++)
    {
        //calc checksum
        CheckSum += *bPtr++;
    }
    //check the checksum
    if(CheckSum != Header.HeaderCS)
    {
        //Checksum Error
        return FALSE;
    }
    //get the data start address
    CurrAdd = Header.DataStartAdd;
    //zero the checksum
    CheckSum = 0;
    //read the data records and write them to the slaves
    for(Index = 0; Index < Header.NumRecords; Index++)
    {
        //read a record
        EepromRead((word)&Data, CurrAdd, sizeof(t_EEpromData) / 2);
        //calculate the checksum
        bPtr = (byte*)&Data;
        for(DataCsIndex = 0; DataCsIndex < sizeof(t_EEpromData); DataCsIndex++)
        {
            CheckSum += *bPtr++;
        }
        //check if the chip address is broadcast
        if(Data.ChipNo == BROADCAST_ADD)
        {
            //when working in I2C then loop over all chip addresses
            //and write the data to all
        }
        else
        {
            //write the data to the slaves
            RotemWrite(Data.ChipNo, Data.RamAdd, (word)&Data.Data, 1);
        }
        //advance the current EEprom address
        CurrAdd += sizeof(t_EEpromData);
    }
    //read a the data CS to the CurrAdd var to save space
    EepromRead((word)&CurrAdd, CurrAdd, 1);
    //check if the checsum matches
    if(CheckSum != CurrAdd)
    {
        //Data Checksum Error
        return FALSE;
    }
    //return success
    return TRUE;
}
```

EEPROM Programming

EEPROM programming can be performed either by on-board programming (after the EEPROM is assembled on the boards) or by the universal EEPROM programmer that supports the Serial Peripheral Interface (SPI) EEPROMs before assembling the EEPROM on the board. Typically, in order to program the AT25080A/160A/320A/640A, two individual instructions must be executed:

- The device must be 'write enabled' via the WREN instruction.
- Only then can a Write (WRITE) instruction be executed.

For detailed information on the EEPROM programming process, refer to the ATMEL AT25080A/160A/320A/640A data sheet. The Microsemi's™ dedicated SW GUI, Cat. (No. SS-0070-00N.) also supports direct EEPROM programming using an Aardvark I^{2}C/SPI Host adapter connected as described in Figure 4.

Note!
An Aardvark I^{2}C/SPI Host adapter joined by its relevant installation CD must be purchased separately.

On Board Programming

To program the EEPROM directly on board, the PD69012 devices must be held in Reset state during the programming phase to avoid bus collisions on the SPI bus.

Figure 4 shows the interconnection between the EEPROM device and the Aardvark I^{2}C/SPI Host adapter. In the same way, each EEPROM programmer or an SPI adapter can be utilized to program the EEPROM device while on-board.

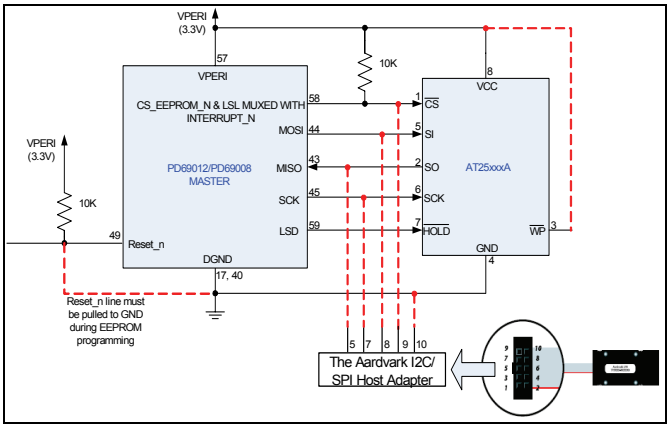

Figure 4: Interconnection between EEPROM and AADVARK SPI Host Adapter

Revision History

Revision Level / Date	Para. Affected	Description
0.1 / 02 Feb. 09	-	Initial Release – Preliminary version
0.2/ 28 July 09	-	Auto Mode System with Host option update, formatting, Preliminary version

For support contact: sales_AMSG@microsemi.com

Visit our web site at: *www.microsemi.com* **Catalog Number: 06-0029-081**