



Advanced
Micro
Devices

Advanced In-circuit Programming Guidelines for MACH[®] 3 and 4 Devices

Application Note

Advanced In-circuit Programming Guidelines for MACH[®] 3 and 4 Devices



Advanced
Micro
Devices

Application Note

by Kenneth Cuy, Applications Engineer, Programmable Logic Division and
David Stoenner, Member of the Technical Staff, Field Applications

INTRODUCTION

This application note serves as a guideline for using the JTAG interface on selected MACH 3 and 4 devices to perform in-circuit programming via a microcontroller. Although this application note will concentrate on an in-circuit programming example with an AMD Am29240[™] RISC microcontroller, the information contained within can be extended to other such applications.

An additional application note entitled *Introduction to JTAG and Five-Volt Programming with MACH[®] 3 and 4 Devices* is available to provide the basic knowledge required to understand the topics of JTAG and five-volt, on-board programming.

System Configuration

The parallel port of a standard PC was chosen as the initial tool to interface with the JTAG port for programming control. The ubiquitous availability of PCs provide a universal development environment for the MACH 3 and 4 devices that support JTAG.

The standard parallel port on a PC is composed of two register addresses in the I/O space of the PC. The two ports are the data port and the control port. The control port consists of two signals; Control which writes to the address and Status which reads to the address.

The six JTAG lines were placed on the parallel port in the following manner. TMS, TCLK and TDI are positive true, output only lines and thus were chosen to be put on the data port. As a convenience for the programmer, the data port is a positive true interface. TRST* and ENABLE* are negative true, output only lines and thus were chosen to be put on the control port. Besides being a negative true interface, the control port is always reset to a known value on power-up. TRST* and ENABLE* were selected to be Strobe and Printer Initiate signals respectively.

The only input needed is for TDO, which was selected as input on the Status Register bit D7, the Printer Busy bit.

The interfacing PAL[®] device, that is used for in-circuit programming, was developed to mimic the parallel port, except that only the bits needed were implemented. This allows the entire control to be put in one PAL device for simplicity, cost, and board space considerations. The example design uses a PALCE22V10 to implement the in-circuit programmability interface for the 29240

microprocessor. The design file for the 22V10 is included in Appendix A.

Interfacing PAL Device

The PALCE22V10 contained in the file JTAG.V10 has been designed to look like a standard microprocessor peripheral using a Chip Select (\overline{CS}), I/O Read (\overline{RD}), I/O Write (\overline{WR}) and A0. The 22V10 is composed of two sections, one to support the data port and the other to support the control and status port. The A1 bit is used in the Am29240. A higher order chip select (PIACS0) is provided with A23 – A0. This allows the chip select to occupy 16 megabytes of address space. A1 is used in the equations so the MACH device can be accessed independent of the JTAG controller.

In the Am29240, the two registers (normally byte oriented) are configured on word (32 bit) boundaries for both the Am29240 hardware and the C software. Therefore, the A0 bit in the JTAG.V10 design is connected to the Am29240 address bit A2, and A1 in the 22V10 is connected to A3 of the Am29240.

The PALCE22V10 has only one clock input (pin 1) which can be used to clock any of the macrocells within the PALCE22V10. However, two registers in the design require different clocks. To solve this problem, the clock pin is used to clock the data register and the control register is implemented as transparent latches arranged in a master/slave configuration. The equation WR_CLK is used to generate the clock for the data register and is externally looped back to pin 1. The equation for WR_CLK is simply the chip select ANDED with I/O write at the correct address. D0, D1 and D2 are then registered into TDI, TCLK and TMS respectively. This is the same bit order that is used in the PC printer port configuration. These outputs are controlled by the term TRI_DISABLE which in turn is controlled by the parallel port cable. If the cable is plugged in then the outputs of the PALCE22V10 are disabled to prevent output contention. This arrangement can even be used in a manufacturing system for both development as well as in field updates.

The TRST* signal is generated from a reset input as well as the master latch called TRST_LAT. The input pin RESET will not only reset all the registers and latches but will also enable TRST* on the interface to reset the JTAG state machine in the selected MACH 3 or 4 device. In the design file, the latch equations are surrounded by MINIMIZE_OFF and MINIMIZE_ON. This is done because of the cross terms in the latch equations

that are necessary to hold the output stable as the latch enable term goes false.

The slave latches are controlled only by chip select (CS) so any activity with the interfacing PAL device will hold TRST* and ENABLE* stable.

The remainder of the PALCE22V10 design is very straight forward in its implementation and can be followed in the design file.

System Dependent Source Code

The software written for the Am29240 interface consists of a group of low level drivers written in pseudo ANSI C for the 29K™ family and is included in Appendix B. These drivers are the only ones that change from system to system. If a different interface was chosen for the JTAG port, these drivers are all that is needed to be re-written for the target system. These drivers are written so that they can directly interface to the C compiler for the 29K family and adhere to the 29K calling convention.

Please contact your local sales office for details on a supplement entitled *MACHPRO™ Downloading Software: C Source Code* which contains the C source code for AMD's MACHPRO programming software. MACHPRO uses the JTAG circuitry on selected MACH devices to program and verify the devices via the PC parallel port. The required drivers listed below need to be integrated with the existing MACHPRO software so that programming may now be performed via a microcontroller.

Required Drivers

Listed below are all of the low level drivers necessary to implement in-circuit programming with the selected JTAG compatible MACH devices. An effort was made to keep the drivers simple and easy to embed for the user.

■ _JTAG_reset_interface()

This routine is called at the beginning of the main program and establishes the JTAG interface to a known state.

This state is:

ENABLE*	FALSE (Logic High)
TRST*	FALSE (Logic High)
TMS	FALSE (Logic Low)
TCLK	TRUE (Logic High)
TDI	FALSE (Logic Low)

■ _JTAG_reset()

This routine will pulse the TRST* pin for a reset pulse

■ _JTAG_pgm_pulse(pulse_width)

This routine will pulse the ENABLE* pin for the duration of the input variable pulse_width in milliseconds. For the 29K there is a system call to the Host Interface (HIF) that will return a clock with a resolution of 1 millisecond. So this routine will set the ENABLE* pin true and then call the clock routine once to establish a starting time and then it will loop calling the clock routine again until the correct time has elapsed. It will then set the ENABLE* pin false and return.

■ _JTAG_pgm_set(value)

This routine will set the ENABLE* pin to the value specified in the D3 bit position of the input variable.

■ _JTAG_shift(TDI_value, TMS_value)

This routine will shift out one bit on to the TDI line and at the end will read the TDO input and return this value. The routine will first set TCLK low, TDI to the value specified in the TDI_value variable and TMS to the value specified in the TMS_value variable. Then the TCLK pin will be set high and then set low and the D7 bit will be read and right justified in the return register.

■ _setup()

This routine simply sets up the PIA port of the Am29240 for 3 clock cycle wait state operation. This is also compatible with the Am29200™.

REFERENCES

Introduction to JTAG and Five-Volt Programming with MACH® 3 and 4 Devices

MACH® 3 and 4 Family Data Book

MACHPRO™ Programming Software Manual

Am29240™, Am29245™, and Am29243™ RISC Microcontrollers User's Manual and Data Sheet

IEEE Standard 1149.1–1990

IEEE Standard 1149.1–1990 Supplement A



PALASM® Design Files

```

;PALASM Design Description

;----- Declaration Segment -----

TITLE      JTAG CONTROL PAL FOR MACH 3XX AND MACH4XX PAL FAMILY
PATTERN    JTAG.V10
REVISION   A
AUTHOR     DAVID STOENNER
COMPANY    ADVANCED MICRO DEVICES

DATE       06/21/93

CHIP       _jtag PAL22V10

;----- PIN Declarations -----

PIN 1      CLK                      ;
PIN 2      /CS                      ;
PIN 3      /RD                      ;
PIN 4      /WR                      ;
PIN 5      A0                      ;
PIN 6      A1                      ;
PIN 7      D0                      ;
PIN 8      D1                      ;
PIN 9      D2                      ;
PIN 10     D3                      ;
PIN 11     /RESET                  ;
PIN 12     GND                    ;
PIN 13     /TRI_DISABLE            ;
PIN 14     D7                      ;
PIN 15     /PRM_EN                 ;
PIN 16     TRST_LAT                ;
PIN 17     PRM_EN_LAT              ;
PIN 18     /TRST                   ;
PIN 19     TDO                    ;
PIN 20     TDI                    ;
PIN 21     TMS                    ;
PIN 22     TCLK                   ;
PIN 23     /WR_CLK                 ;
PIN 24     VCC                    ;
NODE 1     GLOBAL                  ;

;----- Boolean Equation Segment -----

EQUATIONS

GLOBAL.RSTF = RESET

MINIMIZE_OFF

```

```
TRST =    RESET
        + TRST_LAT*/CS*/RESET
        + TRST*TRST_LAT*/RESET
        + TRST*CS*/RESET

PRM_EN =    PRM_EN_LAT*/CS*/RESET
            + PRM_EN*PRM_EN_LAT
            + PRM_EN*CS*/RESET

TRST_LAT =    CS*WR*A0*D0*/A1*/RESET
              + TRST_LAT*/(CS*WR*A0*/A1)*/RESET
              + TRST_LAT*D0*/RESET

PRM_EN_LAT =    CS*WR*A0*/A1*D3*/RESET
                + PRM_EN_LAT*/(CS*WR*A0*/A1)*/RESET
                + PRM_EN_LAT*D3*/RESET

MINIMIZE_ON

WR_CLK = CS*WR*/A0*/A1*/RESET

; NOTE PIN 23 ( WR_CLK ) IS EXTERNALLY CONNECTED TO PIN 1 ( CLK ) FOR
; THE CLOCKED REGISTER.

TDI := D0
TCLK := D1
TMS := D2
D7 = TDO

D7.TRST = CS*RD*/A1*A0*/RESET
TRST.TRST = /TRI_DISABLE*/RESET
TDI.TRST = /TRI_DISABLE*/RESET
TCLK.TRST = /TRI_DISABLE*/RESET
PRM_EN.TRST = /TRI_DISABLE*/RESET

;-----
```



Low Level Drivers

```
pragma On(Pointers_compatible_with_ints);

volatile int *Data_Reg ;
volatile int *Command_Reg ;
volatile int *Status_Reg ;
volatile int *PIA_Control;

#define TRST 0x1
#define PGM_EN 0x8
#define TDI 0x1
#define TCLK 0x2
#define TMS 0x4

extern clock();

void JTAG_reset_interface()
/* We will give the interface a reset pulse and then do 5 TCLK with TMS
 = 1 which will reset the state machine to reset if the reset pulse
 did not work. It will then leave the interface with the command
 register = 0 and the data register = tclk = 0 tdi = 0 and tms = 1. */
{
    *Command_Reg = TRST;
    *Command_Reg = 0;
    *Data_Reg = TMS;
    *Data_Reg = TMS | TCLK;
    *Data_Reg = TMS;
    *Data_Reg = TMS | TCLK;
    *Data_Reg = TMS;
    *Data_Reg = TMS | TCLK;
    *Data_Reg = TMS;
    *Data_Reg = TMS | TCLK;
    *Data_Reg = TMS;
    *Data_Reg = TMS | TCLK;
    *Data_Reg = TMS;

    return;
}

void JTAG_reset()
{
    *Command_Reg = TRST;
    *Command_Reg = 0;

    return;
}

void JTAG_pgm_pulse(w)
```

```
    int w;
{
    int start;

/* Now that the program enable is on we will start the timeout in
milliseconds for the pulse width. A system call to clock ( HIF
service call 273 ) will return the time in milliseconds in gr96, so
we will just do a compare to the pulse width value and loop till the
time has passed. */

    start = clock();
    *Command_Reg = PGM_EN;

/* now do another system clock call and start the timeout loop */

    while(clock()-start < w){;}
    *Command_Reg=0;

    return;
}

void JTAG_pgm_set(in)

int in;
{
    *Command_Reg = (in & 0x1)<<3;

    return;
}

int JTAG_shift(tdi,tms)

    int tms,tdi;

{

    int temp;
    temp = ( tms & 1 ) << 2 | ( tdi & 1 );
    *Data_Reg = temp;
    *Data_Reg = temp | TCLK;
    *Data_Reg = temp;
    return(( *Status_Reg & 0x80 ) >> 7 );

}

void setup()

/* This routine sets up the PIA controller on the 292XX microcontroller
to the needed values. For any race conditions we will set the IO
extend bit and make the interface 3 wait states so that both the
29200 and 29240 behave identical. This port is at 0x80000020. We will
only affect PIACS0 so we will leave the other set up as is. */

{

    Data_Reg = 0x90000000;
    Command_Reg = 0x90000004;
    Status_Reg = 0x90000004;
    PIA_Control = 0x80000020;
    *PIA_Control = ( *PIA_Control & 0x00ffffff ) | 0x83<<24;

    return;
}
```

Copyright © 1994 Advanced Micro Devices, All rights reserved.

AMD, the AMD logo, MACH, PAL and PALASM are registered trademarks of Advanced Micro Devices, Inc.

29K, Am29200, Am29240, Am29245, Am29243, and MACHPRO are trademarks of Advanced Micro Devices, Inc.

Product names used in this publication are for identification purposes only and may be trademarks of their respective companies.