A Tutorial for In-Circuit Programming the ACEx™ Family of Microcontrollers

ACE1001/8001, ACE1101, ACE1202, ACE1502

Fairchild
App Note 2001
February 2003

FAIRCHILD SEMICONDUCTOR®

1.0 Introduction

The ACEx (Arithmetic Controller Engine) family of programmable integrated circuits is designed for applications requiring high performance, low power, and small size. The ACEx family supports in-circuit programming of the code EEPROM array, the data EEPROM array, and initialization registers 1 and 2. In-circuit programming of an ACEx device is accomplished through a 4-pin interface. Data is shifted serially in and out of the device using a 32-bit command/response word. This 32-bit command and response word contains all the necessary information to program data to the EEPROM arrays or the initialization register.

This application note provides a detailed description of the programming interface, the 32-bit command/response word, and the programming procedures. In addition, this application note also provides all timing specifications for the programming signals.

2.0 Interface

The 4-pin interface was designed for simple in-circuit programming. You simply shift data serially in and out of the ACEx device through the following externally controlled signals (see Figure 1):

- **LOAD** control signal
- **CLOCK** signal
- serial data **SHIFT_IN** input signal
- serial data **SHIFT_OUT** output signal

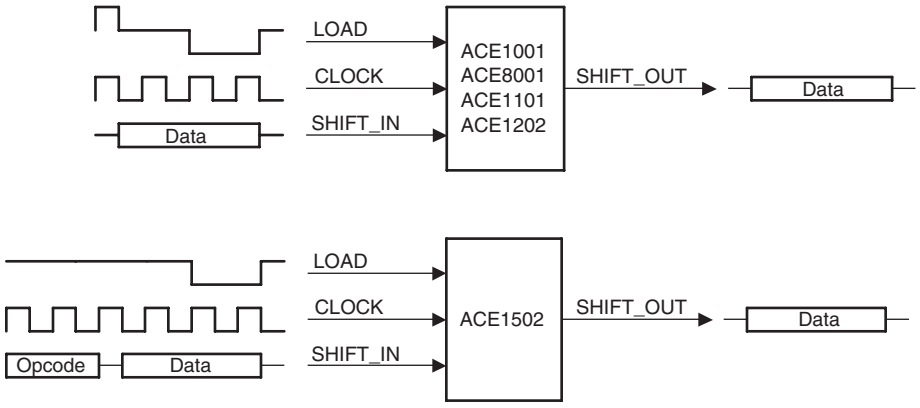
The four interface pins are multiplexed with a particular I/O pin yielding a separate device pinout while in programming mode. Table 1 describes the programming mode device pinout.

Note: I/O pin G5 must either float or be held high on the board in order to read data from SHIFT_OUT. See section 6.0 on gang-programming for more information about the use of G5.

Table 1. Programming Signals

Interface Pin	I/O Pin
LOAD	G3
CLOCK	G1
SHIFT_IN	G4
SHIFT_OUT	G2
VCC	
GND	
NC or High	G5

Figure 1. Programming Interface Diagram



3.0 Command and Response Word

The 32-bit command and response word is shifted serially in (SHIFT_IN) and out of (SHIFT_OUT) the ACEx device. When data is being shifted into the device it is referred to as the 32-bit command word. Likewise, when data is being shifted out of the device it is referred to as the 32-bit response word. Bit 31 of the command word is the first bit to be shifted into the device following all other (30, …, 0) bits. As bit 31 of the command word is shifted in, bit 31 of the response word is being shifted out simultaneously. (The response word is the ACEx responding to the 32-bit command word shifted previously.)

The 32-bit command/response word uses a simple format consisting of two control bits, a read/write bit, address bits, and data bits. See Table 2 for the details displayed in a bit-oriented format. Table 3 displays the same information in a byte-oriented format.

The two control bits (bit28, bit29) select the memory. The read/write bit (bit24) selects whether you are reading or writing to the memory address.

The address bits specify the memory address to read from or write to. The memory address is 10 bits wide for ACExx01 devices and 11 bits wide for ACExx02 devices. To program the 12-bit code space addresses, use the lower 10/11 bits of the memory-mapped addresses.

Note: Currently, the 12-bit address may be used without masking to 10 or 11 bits. Future devices may not work this way.

When performing a write, the data bits specify the data to write to the memory address in both the command and the response word. However, when performing a read, the data bits in the command word contain different values than response word. When performing a read, the data bits in the command word should be *zeros,* and the data bits in the response word contain the data of the memory at the specified address location.

Table 2: 32-bit Command and Response Word, Bit Layout

Bit #	Input Command Word	Output Response Word
31-30	Must be set to 0	X
29	Set to 1 for addresses < 0xFF, otherwise 0	X
28	Set to 1 for addresses > 0xFF, otherwise 0	X
27-25	Must be set to 0	X
24	Set to 1 to read, 0 to write	X
23-19	Must be set to 0	X
18-8	Address of the byte to be read or written	Same as input command word
7-0	Data to be written or zero if data is to be read	Data written or data read at specified address

Table 3: 32-bit Command and Response Word, Byte Layout

Instruction	32-bit Command (MSB to LSB)				Operation
	Byte 1	Byte 2	Byte 3	Byte 4	
Read Code Memory Space	00010001 (0x11)	00000**UUU**	**LLLLLLLL**	**xxxxxxxx**	Read code memory space **U:L** (0 to 0x7FF)[1]. Data will be read at the next 32-bit Read Command (**x** is the value read following a previous Read Command)
Write Code Memory Space	00010000 (0x10)	00000**UUU**	**LLLLLLLL**	**dddddddd**	Write 8-bit data **d** in code memory space **U:L** (0 to 0x7FF)[1]
Read Data Memory Space	00100001 (0x21)	00000000	**bbbbbbbb**	**xxxxxxxx**	Read data memory space **b** (0 to 0xFF, data EEP = 0x40 to 0x7F).). Data will be read at the next 32-bit Read Command (**x** is the value read following a previous Read Command)
Write Data Memory Space	00100000 (0x20)	00000000	**bbbbbbbb**	**dddddddd**	Write 8-bit data **d** in code memory space **b** (0 to 0xFF, data EEP = 0x40 to 0x7F)
Read Initialization Register 1	00100001 (0x21)	00000000	10111011 (0xBB)	**xxxxxxxx**	Read Initialization Register 1 (0xBB). Data will be read at the next 32-bit Read Command (**x** is the value read following a previous Read Command)
Write Initialization Register 1	00100000 (0x20)	00000000	10111011 (0xBB)	**76543210**	Write Initialization register 1 (0xBB): **7+6** = Osc. Option, **5** = Watchdog, **4** = Brownout, **3** = Varies[2], **2** = Upper Block Disable, **1** = Write Disable, **0** = Read Disable.
Read Internal Oscillator Register	00100001 (0x21)	00000000	10111100 (0xBC)	**xxxxxxxx**	Read Internal Oscillator trimming register (0xBC) data will be read at the next 32-bit Read Command (**x** is the value read following a previous Read Command)
Write Internal Oscillator Register	00100000 (0x20)	00000000	10111100 (0xBC)	**dddddddd**	Write Internal Oscillator trimming register (0xBC) with **d**

[1]The program code address space is 0-0x3FF on ACExx01 devices and 0-0x7FF on ACExx02 devices. During programming, the 12-bit memory-mapped addresses are masked to their lower 10 or 11 bits, respectively.

[2]See the data sheet for details on bit3 of the device's initialization register. It can be used for brown-out or low-battery-detect settings. It may be necessary to preserve bit 3's setting when writing a new value to the register.

4.0 Programming Procedures

There are several basic steps to program memory in an ACEx device. The process includes placing the device in programming mode, clocking the 32-bit command word into the device, and providing two additional clocks with the control signal low to perform the read or the write.

The following steps are required to program an ACEx device's memory:

Step 1: Put the device in programming mode. There are two ways to do this, depending on which ACEx device you are using:
- – **ACE1001/8001/1101/1202:** Supply a single 12V supervoltage pulse to the LOAD signal
- – **ACE1502:** Shift an opcode into the device during the power-on-reset process with LOAD at Vcc. (ACE1502 devices are 3.3V devices and will be damaged by a 12V supervoltage pulse.)

Step 2: Set LOAD to Vcc.

Step 3: Shift in bit 31 of the 32-bit command word through the SHIFT_IN pin.
- – Continue shifting in bits 30 through 0.

Step 4: Supply a CLOCK pulse.

Step 5: Repeat Steps 3 and 4 until all 32 bits of the command word are shifted into the device.
- – If the previous command word contained the read command, poll SHIFT_OUT at t_{ACCESS} after the rising edge of CLOCK to obtain the 32-bit response word.

Step 6: Set LOAD to 0V.
- – SHIFT_OUT goes to Vcc.

Step 7: Supply two CLOCK pulses to perform the read or write.
- – When performing a write, SHIFT_OUT must be at 0V before the second CLOCK pulse is provided. SHIFT_OUT is now the BUSY signal during the write process.

Step 8: Poll SHIFT_OUT for the READY signal.
- – After the second CLOCK pulse is supplied and the write is completed, SHIFT_OUT returns to Vcc. SHIFT_OUT is now the READY signal. The READY signal must be high before LOAD can return to Vcc.

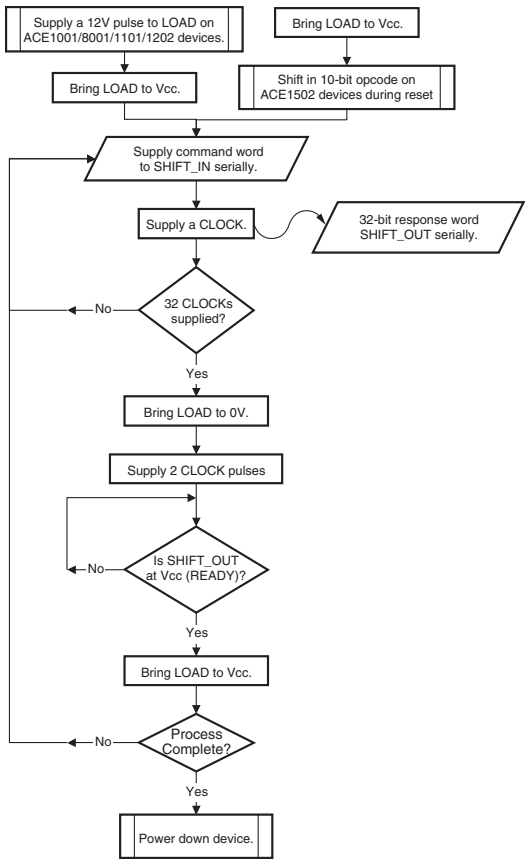
Step 9: Repeat Steps 2 through 7 until all bytes are read from or written to memory.

Step 10:Exit programming mode by powering down the device.

Note: On reads, the data shifted out comes from the address shifted in on the *previous* command word. For example: In order to read two bytes, three command words need to be shifted into the device. The first two command words contain the addresses to be read, and the third is a dummy using zero for the address. The first data byte read will return random data and can be discarded, the next two will be the data read from the addresses in the first two command words.

Figure 2. Process Flow-diagram

Figure 2 provides a flow diagram of the general process for the entire ACEx family. Please refer to the specific data sheet for details of a particular device.

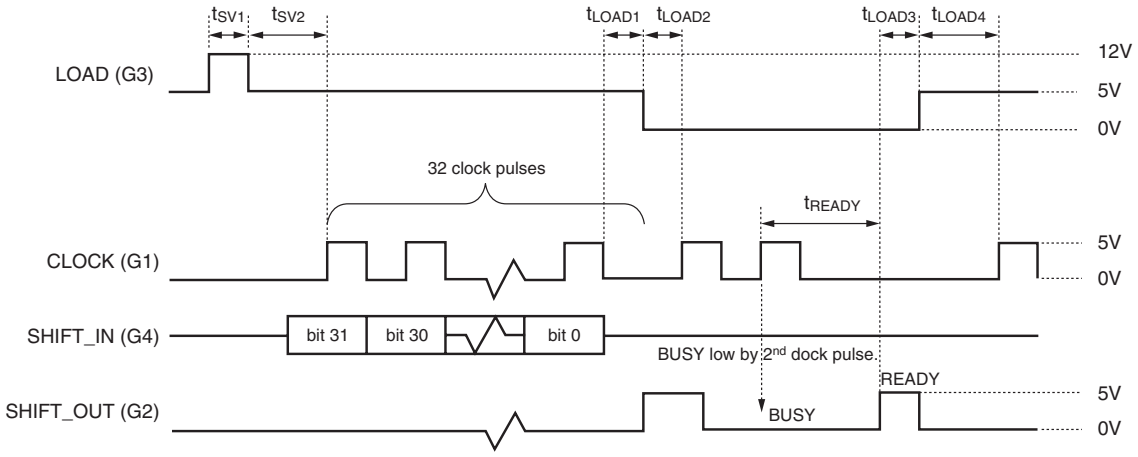


5.0 Timing Specification

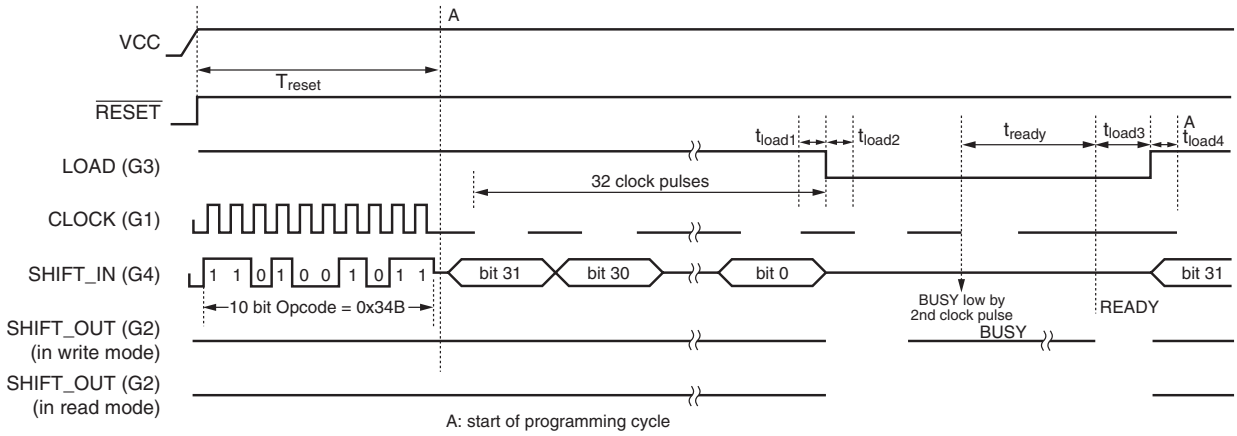
Figure 3 shows the programming protocol, including the specific timing parameters defined in Table 4.

Figure 3. Programming Protocol

Supervoltage Method: ACE1001/8001/1101/1202



Opcode Method: ACE1502



Figures 4 and 5 show the serial data timing, including the specific timing parameters defined in Table 4.

Figure 4. ACE1001/ACE1101 Serial Data Timing

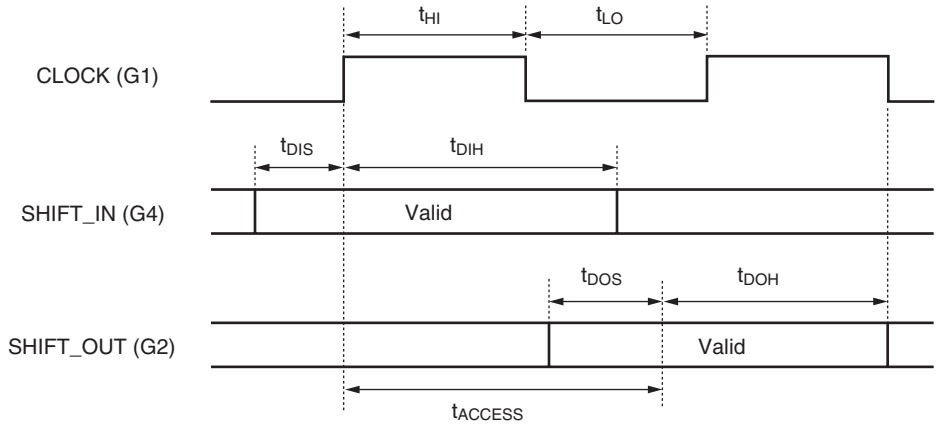


Figure 5. ACE1202/ACE1502 Serial Data Timing

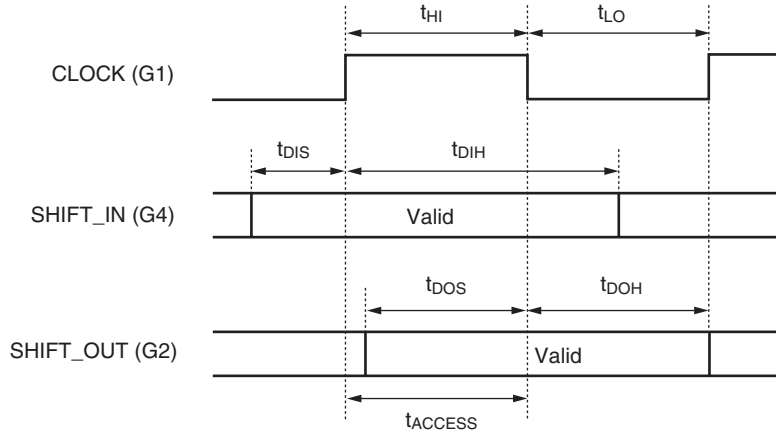


Table 4: Programming Electrical Characteristics and Timing Parameters

Parameter	Description	Min.	Max.	Units
t_{HI}	CLOCK high time	500	DC	ηs
t_{LO}	CLOCK low time	500	DC	ηs
t_{DIS}	SHIFT_IN setup time	100		ηs
t_{DIH}	SHIFT_IN hold time	100		ηs
t_{DOS}	SHIFT_OUT setup time	100		ηs
t_{DOH}	SHIFT_OUT hold time	100		ηs
t_{ACCESS} (ACE1101)	SHIFT_OUT access time	900		ηs
t_{ACCESS} (ACE1202)	SHIFT_OUT access time	500		ηs
t_{SV1}, t_{SV2}	LOAD supervoltage timing*	50		µs
t_{LOAD1}, t_{LOAD2}, t_{LOAD3}, t_{LOAD4}	LOAD timing	5		µs
$V_{SUPERVOLTAGE}$	Supervoltage level*	11.5	12.5	V
T_{RESET} (ACE1502 only)	Power-On Reset	3.2	4.5	ms

*Supervoltage should not be used in an ACE1502

6.0 Gang Programming

This section describes the mass-production programming process. Fairchild uses this special method during wafer-sort to test a large number of ACEx units and to optimize the testing time.

The programming procedures for gang programming are generally the same as those described earlier in this application note. The I/O pin G5 is used as a chip select. When G5 is set high, SHIFT_OUT is enabled—allowing the device to be read. When G5 is set low, SHIFT_OUT is disabled and the device cannot be read. One time-saving strategy is to do all the writes in parallel to several devices at the same time. Then, using a Mux, the parts are read-enabled one-by-one by setting G5 to Vcc. Each device is read and the write data verified. Since SHIFT_OUT is disabled and cannot be polled to detect when the write has completed, the process must wait a fixed delay time. To ensure that all devices have completed the process, this time should be equal to the maximum write time.

Note: SHIFT_IN is not disabled with G5 and all connected parts will be programmed.

The following steps show how to use a supervoltage pulse to gang-program devices:

Note: Do not use supervoltage pulses with the ACE1502 device. Doing so will damage the device.

1. Put the devices in programming mode is by bringing the signal LOAD to +12V after having applied Vcc (4.5 to 5.5V).
2. Set G5 low on all connected devices to disable SHIFT_OUT (deselect external Mux.)
3. Program the desired locations without verifying the READY status on SHIFT_OUT. Wait 10ms before programming the next location; all the devices connected to the bus will be programmed simultaneously.
4. Verify each device by bringing the respective G5 high. (Refer to the interface shown in Figure 6 to select the device no. 1 the mux selection pins must be 0000 to select the output Y0 and so on).

To gang program ACE1502 devices, put them in programming mode with the opcode method after applying Vcc (3.3V). The opcode method is described in “4.0 Programming Procedures” of this application note. Then, follow steps 2 through 4 listed above.

Figure 6. Gang Programming Interface:

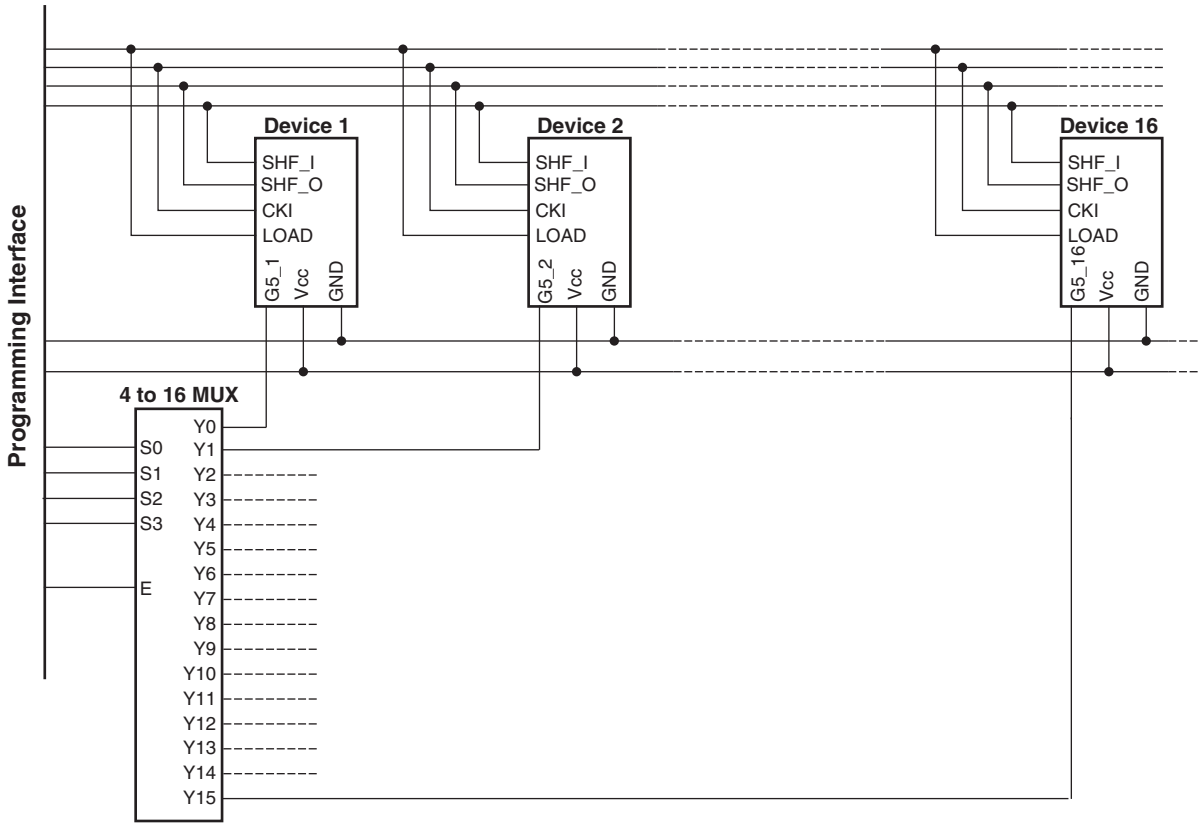
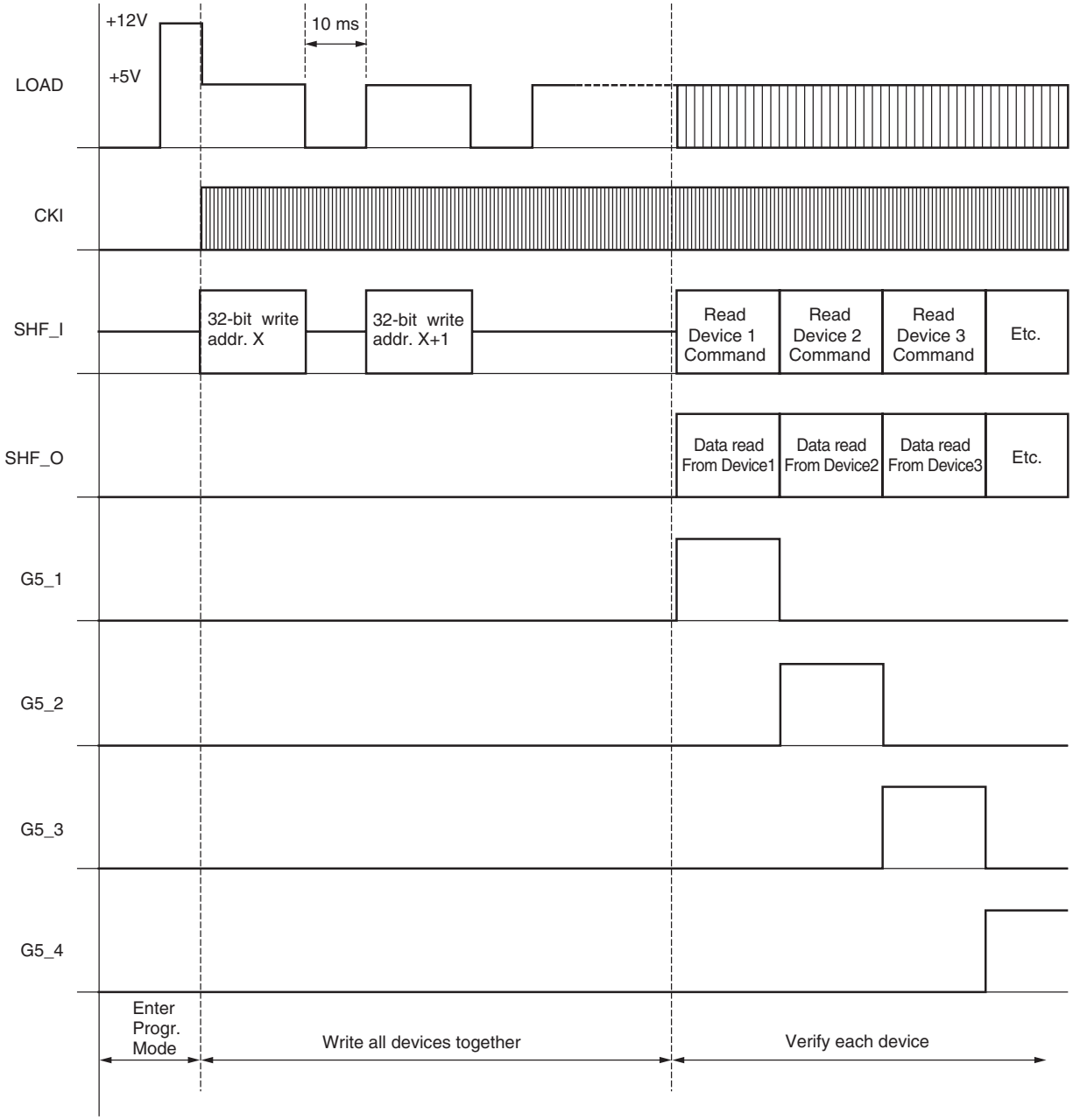


Figure 6 – Program-by-sixteen interface

Figure 6 shows how a simple hardware interface can program sixteen devices simultaneously. A digital (4 to 16) multiplexer minimizes the number of wires needed to select a single device. If the signals are properly buffered, the number of devices that can be programmed in parallel is virtually unlimited.

Gang Programming Protocol:



7.0 Trimming the Internal Oscillator

Initialization Register 2 is used to calibrate the internal clock oscillator. By default the internal clock is trimmed to 2MHz during production. This is the recommended speed for operating the device. Just as some PC users choose to “overclock” their CPUs for added performance, some ACEx device users need higher clock speeds for their applications. For example, the new ACE1502-based emulator board v2.0 has two ACE1502 BIOS devices. One is dedicated to emulation and the second to programming. Both of these devices are trimmed to 8MHz to enhance the board's performance.

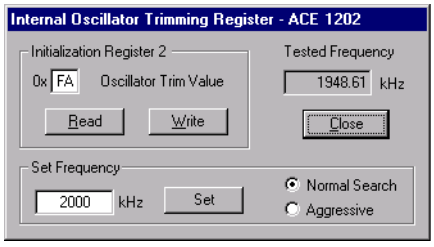
There are a number of facts about the ACEx devices’ internal oscillators to know before experimenting with clock settings. When the register is set at production, there are different standards for the different devices in the family. The ACE1101/1202 devices are specified as set with ±10% accuracy. The ACE1001/8001 devices have an improved accuracy of ±4%, and the ACE1502 is extremely accurate at ±2%. Experience has shown the ACE1502 to be reliable at the 8MHz attained through the trim register. Lab tests have run the device in the 28-32MHz range when externally clocked. The rest of the family should not be expected to run above the 4-5MHz range, although individual parts may run faster.

The internal clock is affected by temperature and voltage extremes. Please refer to the data sheets for test information on how the individual parts behave in the extremes of their operating ranges. Testing determined it safe to run the emulator board devices at higher clock speeds, because their environment is normally room temperature with a constant voltage.

Various resistors are connected or disconnected to the individual bits of the trim register. This tends to make the value of the register to have a non-linear relationship with the clock speed.

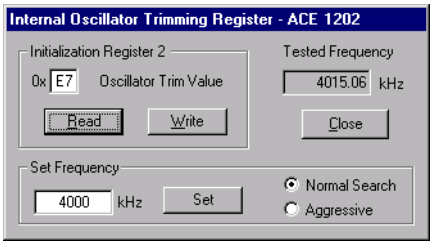
You may wish to experiment with the Emulator board v2.0; trimming a sample of parts of the type chosen for the application. The following is an example of how to use the Emulator software to trim the oscillator.

The clock oscillator trimming feature is found in the “Advanced Programmer” dialog of the Emulator software v8.0. Click on the “Oscillator” button to open this dialog:

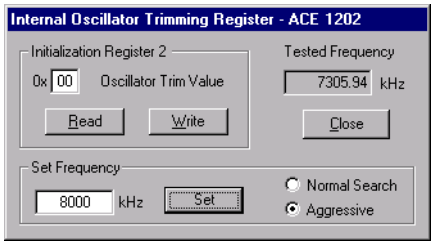


In this example, an ACE1202 has the trim register set to 0xFA. Testing found the clock trimmed to approximately 1949kHz; indicating a 2.6% error from the desired 2000kHz.

Next, trim the part to 4MHz.



The trim utility found the setting 0xE7 to be the best with a measured frequency of approximately 4015kHz. Now the error dropped to 0.375%. In this example, the device was given an 8MHz request to find the highest speed at which this device will perform the calibration test.



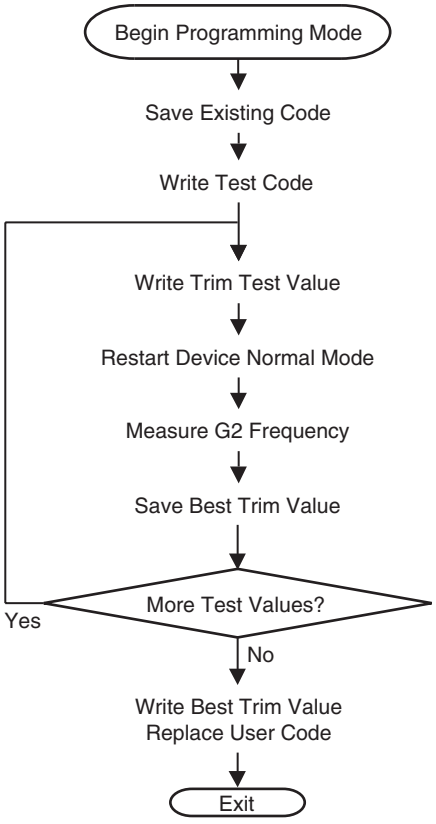
Even though an ACE1202 should not be expected to run above 5MHz, this particular device ran the test at 7.3MHz. Notice the “Aggressive” option is selected. The “Normal Search” tries a reasonable range of values found to result in the speed requested. The test process runs the part at each register setting and measures its timing. Because many settings may cause the part to not run at all, the process has a timeout period. To give the utility better performance, only a range expected to produce success is tried. This reduces time spent waiting for a series of timeouts with unreasonable settings. To remove any doubt that a “good” setting was not tried, the “Aggressive” search option will try every setting from 0 to 0xFF. However, testing has shown that with current devices the “Normal Search” will find the same setting and find it quicker than the “Aggressive” search option.

The table below shows the ranges used by the utility: (Simplified and condensed for display here, the actual table has different breakpoints for each device type.)

	< 1900kHz	1900-2999kHz	3000-3999kHz	4000-5000kHz	> 5000kHz
ACE1001 / 8001	0xE0 - 0xFF	0x83 - 0xF6	0x7E - 0x92	0x2E - 0x7F	0x00 - 0x69
ACE1101 / 1202	0xF0 - 0xFF	0xEC - 0xFC	0xD5 - 0xF5	0xC4 - 0xE6	0xC4 - 0xE6
ACE1502	0xCB - 0xFF	0x8D - 0XF8	0x80 - 0xA7	0x77 - 0xAD	0x00 - 0x7F

To set the oscillator trim register during in-circuit programming, a test method needs to be developed to be sure the desired clock setting is achieved. The steps below describe the procedure used by the trimming utility in the Emulator board. A test program is written to the first 16 bytes of code-space that generates a 1kHz square wave on G2.

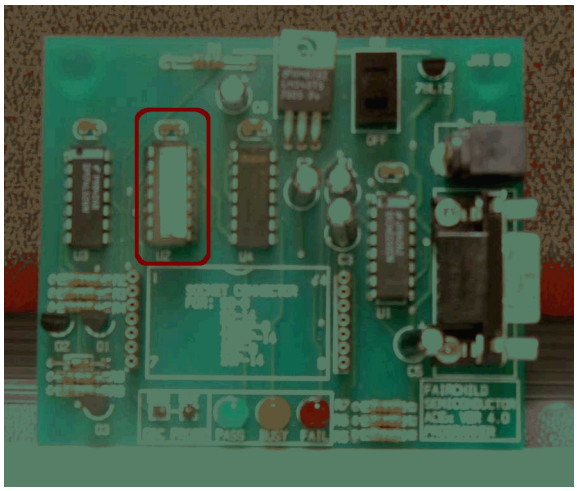
1. Reset the device and place in programming mode
2. Save existing code in the first 16 bytes of the device
3. Write the test code to code space in the device
4. Write a test value to the oscillator trim register
5. Power off the device to exit programming mode
6. Power on the device to start in normal mode
7. Measure the frequency of the wave from G2
8. Save the trim register value with the best measurement
9. To test other values, reset device and place in programming mode. Go to step 4 and repeat as needed
10. Write best value to trim register
11. Replace the user code to code space and exit



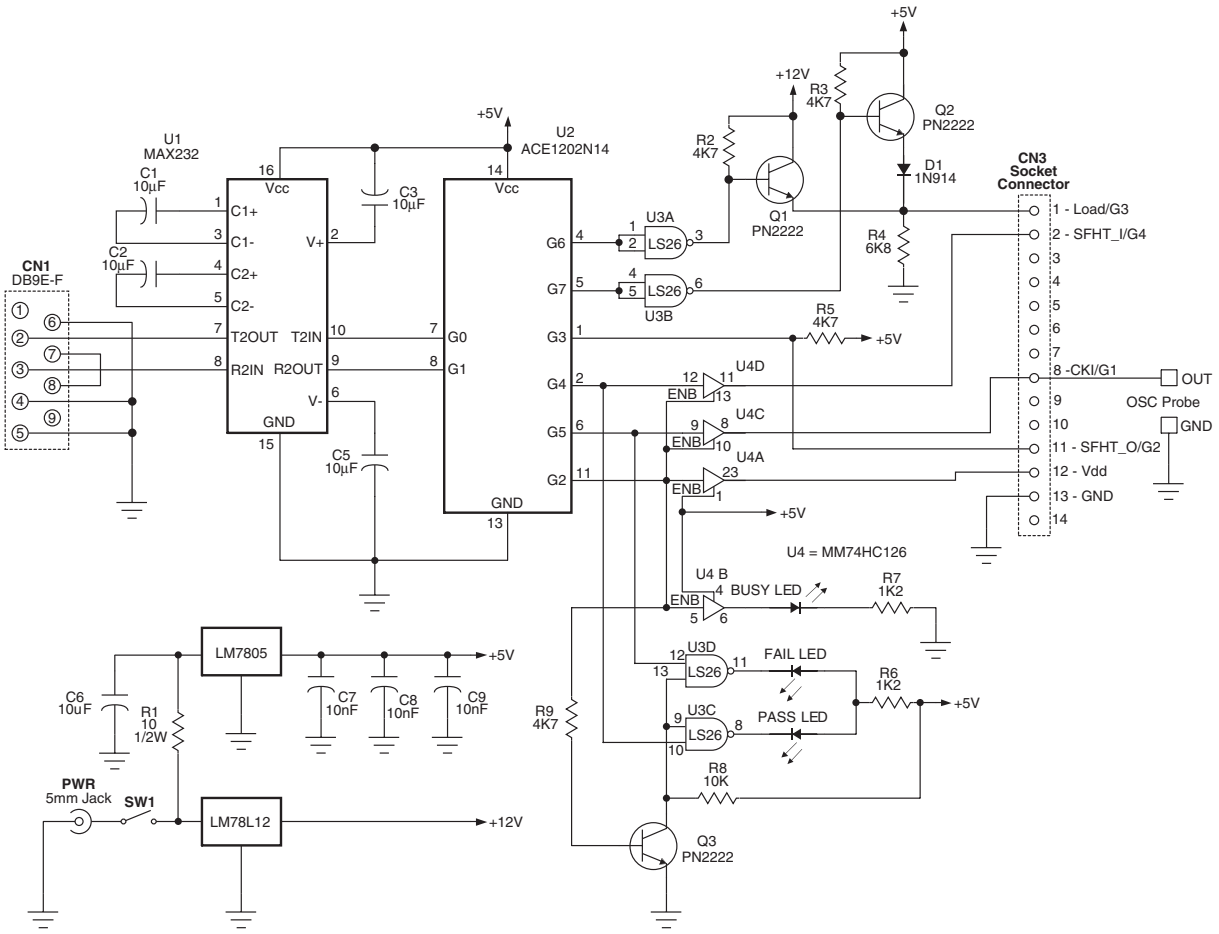
8.0 Example Implementation

An example in-circuit programmable board was constructed using a 1999 Fairchild ACEx Programmer board (ACESTART), v4.0. The original BIOS device, labeled U2 (circled), was removed from the board and replaced with a 14-pin socket. A new, 14-pin DIP, ACE1202 device is used in this socket. This new device is programmed with the assembled code from the InCircuit.asm source file attached to this application note. Devices may be programmed using an adapter socket from any Fairchild Programmer or Emulator board. A cable can be used to connect the board to the target application circuit.

Example in-circuit programmable board:

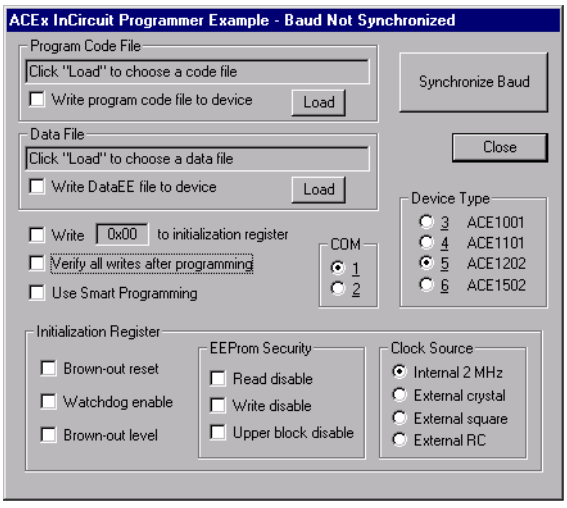


ACEx Programmable Board Schematic:

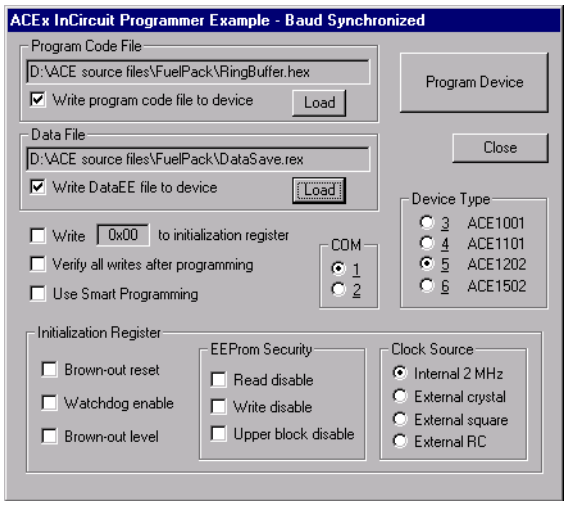


A Windows®-executable file (ACExInCircuitProgrammer.exe) is used to communicate with the board. This executable and all the source files for the Visual C++® 6.0 project are included with this application note. This program only programs the code, DataEE and the initialization register. It does not read back and display the existing contents of the device.

The first menu in this application displays a message indicating the baud has not been synchronized since the board was reset. Click on “Synchronize Baud” to synchronize at 9600 baud. COM1 and COM2 are the only choices available, but these choices can be extended by modifying the included source code.



If the software and the board agree that the baud rate is synchronized, the button will change to “Program Device” and the title bar will reflect that the baud has been synchronized.



To program the device, follow these steps:

1. Select the location of the code and data files from the menu. Click both “Load” buttons.
2. Check the appropriate boxes for the initialization register settings.
3. Click the “Program Device” button to begin programming.

The “Smart Programming” option tells the programmer to read each byte first and if the new value is the same, skip the EEPROM write to save time. The board BIOS code uses a detection method that tries to place the device in programming mode first with the ACE1502 procedure. If this is successful, the 12V supervoltage pulse is skipped. This is to prevent damaging an ACE1502 device in the event that an incorrect device-type was selected. This example implementation does not adjust Vcc to 3.3V for the ACE1502. An actual implementation will set Vcc for the intended target device.

9.0 Source Files

The source files for the PC application are included in “VisualC_Project.zip.” The platform used was Microsoft® Visual C++® 6.0.

The compiled executable file is in “ACExInCircuitProgrammerExe.zip.”

The assembler source for ACE1202 is in “InCircuitAsmSource.zip.”

Microsoft®, Windows® and Visual C++® are registered trademarks of the Microsoft Corporation.

Fairchild Semiconductor Americas
Customer Response Center
Tel. 1-888-522-5372

Fairchild Semiconductor Europe
Fax: +44 (0) 1793-856858
Deutsch Tel: +49 (0) 8141-6102-0
English Tel: +44 (0) 1793-856856
Français Tel: +33 (0) 1-6930-3696
Italiano Tel: +39 (0) 2-249111-1

Fairchild Semiconductor Hong Kong
8/F, Room 808, Empire Centre
68 Mody Road, Tsimshatsui East
Kowloon. Hong Kong
Tel; +852-2722-8338
Fax: +852-2722-8383

Fairchild Semiconductor Japan Ltd.
4F, Natsume Bldg.
2-18-6, Yushima, Bunkyo-ku
Tokyo, 113-0034 Japan
Tel: 81-3-3818-8840
Fax: 81-3-3818-8841