This document includes the programming specifications for the following devices:

- PIC17C42
- PIC17C43
- PIC17C42A
- PIC17CR43
- PIC17C44

## 1.0 PROGRAMMING THE PIC17CXX

The PIC17CXX is programmed using the TABLWT instruction. The table pointer points to the internal EPROM location start. Therefore, a user can program an EPROM location while executing code (even from internal EPROM). This programming specification applies to PIC17CXX devices in all packages.

For the convenience of a programmer developer, a “program & verify” routine is provided in the on-chip test program memory space, the program resides in ROM and not EPROM. Therefore, it is not erasable. The “program/verify” routine allows the user to load any address, program a location, verify a location or increment to the next location. It allows variable programming pulse width.

### 1.1 Hardware Requirements

Since the PIC17CXX under programming is actually executing code from “boot ROM,” a clock must be provided to the part. Furthermore, the PIC17CXX under programming may have any oscillator configuration (EC, XT, LF or RC). Therefore, the external clock driver must be able to overdrive pulldown in RC mode. CMOS drivers are required since the OSC1 input has a Schmitt trigger input with levels (typically) of 0.2V<sub>DD</sub> and 0.8V<sub>DD</sub>. See the PIC17C4X data sheet (DS30412A) for exact specifications.

### Pin Diagram

#### 40L PDIP, Windowed CERDIP

The PIC17CXX requires two programmable power supplies, one for V<sub>DD</sub> (2.5V to 6.0V recommended) and one for V<sub>P</sub> (13 ± 0.25V). Both supplies should have a minimum resolution of 0.25V.

The PIC17CXX uses an intelligent algorithm. The algorithm calls for program verification at V<sub>DD</sub> min as well as V<sub>DD</sub> max. Verification at V<sub>DD</sub> min guarantees good “erase margin”. Verification at V<sub>DD</sub> max guarantees good “program margin”. Three times (3X) additional pulses will increase program margin then beyond V<sub>DD</sub> (max.) and insure safe operation in user system.

### PIN DESCRIPTIONS (DURING PROGRAMMING): PIC17C42/42A/43/44

<table>
<thead>
<tr>
<th>Pin Name</th>
<th>Pin Name</th>
<th>Pin Type</th>
<th>Pin Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RA &lt;0:4&gt;</td>
<td>RA &lt;0:4&gt;</td>
<td>I</td>
<td>Necessary in programming mode</td>
</tr>
<tr>
<td>TEST</td>
<td>TEST</td>
<td>I</td>
<td>Must be set to “high” to enter programming mode</td>
</tr>
<tr>
<td>RB &lt;7:0&gt;</td>
<td>PAD &lt;15:8&gt;</td>
<td>I/O</td>
<td>Address &amp; data: high byte</td>
</tr>
<tr>
<td>RC &lt;7:0&gt;</td>
<td>PAD &lt;7:0&gt;</td>
<td>I/O</td>
<td>Address &amp; data: low byte</td>
</tr>
<tr>
<td>MCLR/V&lt;sub&gt;P&lt;/sub&gt;</td>
<td>V&lt;sub&gt;P&lt;/sub&gt;</td>
<td>P</td>
<td>Programming Power</td>
</tr>
<tr>
<td>V&lt;sub&gt;DD&lt;/sub&gt;</td>
<td>V&lt;sub&gt;DD&lt;/sub&gt;</td>
<td>P</td>
<td>Power Supply</td>
</tr>
<tr>
<td>V&lt;sub&gt;S&lt;/sub&gt;S</td>
<td>V&lt;sub&gt;S&lt;/sub&gt;S</td>
<td>P</td>
<td>Ground</td>
</tr>
</tbody>
</table>

Legend: I = Input, O = Output, P = Power
The actual programming must be done with \( V_{DD} \) in the \( V_{DDP} \) range (4.75 - 5.25V).

\( V_{DDP} = V_{DD} \) range required during programming.

\( V_{DD} \) \text{min.} = minimum operating \( V_{DD} \) spec for the part.

\( V_{DD} \) \text{max.} = maximum operating \( V_{CC} \) spec for the part.

Programmers must verify the PIC17CXX at its specified \( V_{DDmax} \) and \( V_{DDmin} \) levels. Since Microchip may introduce future versions of the PIC17CXX with a broader \( V_{DD} \) range, it is best that these levels are user selectable (defaults are ok).

**Note:** Any programmer not meeting these requirements may only be classified as "prototype" or "development" programmer but not a "production" quality programmer.

### 2.0 PROGRAM MODE ENTRY

To execute the programming routine, the user must hold \( \text{TEST} \) pin high, \( \text{RA2}, \text{RA3} \) must be low and \( \text{RA4} \) must be high (after power-up) while keeping \( \text{MCLR} \) low and then raise \( \text{MCLR} \) pin from \( V_{IL} \) to \( V_{DD} \) or \( V_{PP} \). This will force FFE0h in the program counter and execution will begin at that location (the beginning of the boot code) following reset. Execution is forced to Internal mode by overriding the fuse configuration. The code protect bit is not overwritten. The program immediately polls \( \text{PORT RB<7:0>} \) to determine a branch address. Presenting E1h on \( \text{PORT RB} \) will cause the program to jump to and execute the "program/verify" routine.

**Note:** The OSC must not have 72 osc clocks while the device \( \text{MCLR} \) is between \( V_{IL} \) and \( V_{IHH} \).

All unused pins during programming are in high impedance state.

\( \text{PORTB (RB)} \) has internal weak pull-ups which are active during the programming mode. When \( \text{TEST} \) pin is high, Power-up timer (PWRT) and Oscillator Start-up Timers (OST) are disabled.

#### 2.1 Program/Verify Mode

The program/verify mode is intended for full-feature programmers. This mode offers the following capabilities:

a) Load any arbitrary 16-bit address to start program and/or verify at that location.

b) Increment address to program/verify the next location.

c) Allows arbitrary length programming pulse width.

d) Following a "verify" allows option to program the same location or increment and verify the next location.

e) Following a "program" allows options to program the same location again, verify the same location or to increment and verify the next location.

**FIGURE 2-1: PROGRAMMING/VERIFY STATE DIAGRAM**

![Program/Verify State Diagram](image-url)
2.1.1 LOADING NEW ADDRESS

The program allows new address to be loaded right out of reset. A 16-bit address is presented on ports RB (high byte) and RC (low byte) and the RA1 is pulsed (0 → 1, then 1 → 0). The address is latched on the rising edge of RA1. See timing diagrams for details. After loading an address, the program automatically goes into a "verify cycle". To load a new address at any time, the PIC17C4X must be reset and the programming mode re-entered.

2.1.2 VERIFY (OR READ) MODE

"Verify mode" can be entered from "Load address" mode, "program mode" or "verify mode". In verify mode pulsing RA1 will turn on PORTS RB and RC output drivers and output the 16-bit value from the current location. Pulsing RA1 again will increment location count and be ready for the next verify cycle. Pulsing RA0 will begin a program cycle.

2.1.3 PROGRAM CYCLE

"Program cycle" is entered from "verify cycle" or program cycle itself. After a verify, pulsing RA0 will begin a program cycle. 16-bit data must be presented on PORTS RB (high byte) and RC (low byte) before RA0 is raised.

The data is sampled 3 Tcy cycles after the rising edge of RA0. Programming continues for the duration of RA0 pulse.

At the end of programming the user can choose one of three different routes. If RA1 is kept low and RA0 is pulsed again, the same location will be programmed again. This is useful for applying over programming pulses. If RA1 is raised before RA0 falling edge, then a verify cycle is started without address increment. Raising RA1 after RA0 goes low will increment address and begin verify cycle on the next address.

FIGURE 2-2: PIC17C4X PROGRAM MEMORY MAP

*This location does not exist for PIC17C42
3.0 PROGRAMMING SPECIFICATIONS

FIGURE 3-1: PROGRAMMING ROUTINE FLOWCHART

- B port is forced by the part
- B port is tri-state, should be forced by user

Min RA1 high or low = 10 Tcy
FIGURE 3-2: RECOMMENDED PROGRAMMING ALGORITHM FOR USER EPROM

1. Start
   - Load new address
     - Pulse-count = 0
   - Set Vdd = Vdd min
   - Verify blank
     - Blank check?
       - Yes: Load new data
         - Set Vdd = Vdd max
       - No: Issue "Blank check fail" error message
         - More 100 µs programming pulses for margin (Over programming)
         - Set Vdd = Vdd min
         - Verify location
         - Pass?
           - Yes: Program using 100 µs pulse increment pulse-count
           - No: Location fails programming, issue error message "Unable to program location"
           - Pulse-count +2
         - Location fails programming, issue error message "Unable to program location"
         - Issue error message "Fail verify @ Vdd min/max"
2. Verify blank
   - Issue "Blank check fail" error message
3. Load new data
   - Set Vdd = Vdd max
   - Verify location
   - Pass?
     - Yes: Program using 100 µs pulse increment pulse-count
     - No: Location fails programming, issue error message "Unable to program location"
     - Pulse-count +2
   - Location fails programming, issue error message "Unable to program location"
4. Set Vdd = Vdd min
   - Verify location
   - Pass?
     - Yes: Program using 100 µs pulse increment pulse-count
     - No: Location fails programming, issue error message "Unable to program location"
     - Pulse-count +2
   - Location fails programming, issue error message " Unable to program location"
5. Set Vdd = Vdd max
   - Verify location
   - Pass?
     - Yes: Program using 100 µs pulse increment pulse-count
     - No: Location fails programming, issue error message "Unable to program location"
     - Pulse-count +2
   - Location fails programming, issue error message "Unable to program location"
6. Set Vdd = Vdd min
   - Verify location(s)
   - Pass?
     - Yes: Program using 100 µs pulse increment pulse-count
     - No: Location fails programming, issue error message "Unable to program location"
     - Pulse-count +2
   - Location fails programming, issue error message "Unable to program location"
7. Set Vdd = Vdd max
   - Verify location
   - Pass?
     - Yes: Program using 100 µs pulse increment pulse-count
     - No: Location fails programming, issue error message "Unable to program location"
     - Pulse-count +2
   - Location fails programming, issue error message "Unable to program location"
FIGURE 3-3: RECOMMENDED PROGRAMMING ALGORITHM FOR CONFIGURATION WORDS

Start

- Load new address
  - Pulse-count = 0

- Set $V_{DD} = V_{DD\text{min}}$

- Verify blank

  - No: Issue "blank check fail" error message
  - Yes: Set $V_{DD} = V_{DD\text{min}}$

- Load new data

- Set $V_{DD} = V_{DD\text{PP}}$

- Program using 100 μs pulse increment pulse-count

  - Yes: Pulse count < 100
  - No: Verify location for correct data

  - Yes: Pass?
    - No: Location fails programming, issue error message "Unable to program location"

  - No: Set $V_{DD} = V_{DD\text{min}}$

- Set $V_{DD} = V_{DD\text{max}}$

- Verify location(s)

  - Pass? Yes: Set $V_{DD} = V_{DD\text{min}}$
    - No: Issue error message "Fail verify at $V_{DD\text{min/max}}$"

  - No: Pulse count < 100
    - Yes: Set $V_{DD} = V_{DD\text{min}}$
      - Verify location
4.0 CONFIGURATION WORD
Configuration bits are mapped into program memory. Each bit is assigned one memory location. In erased condition a bit will read as ‘1’. To program a bit, the user needs to write to the memory address. The data is immaterial; the very act of writing will program the bit. The configuration word locations are shown in Table 4-3. The programmer should not program the reserved locations to avoid unpredictable results and to be compatible with future variations of the PIC17C4X. It is also mandatory that configuration locations are programmed in the strict order starting from the first location (0xFE00) and ending with the last (0xFE0F). Unpredictable results may occur if the sequence is violated.

4.1 Reading Configuration Word
The PIC17CXX has seven configuration locations (see Table 4-1). These locations can be programmed (read as ‘0’) or left unprogrammed (read as ‘1’) to select various device configurations. Any write to a configuration location, regardless of the data, will program that configuration bit. Reading any configuration location between 0xFE00 and 0xFE07 will place the low byte of the configuration word (see Table 4-2) into PAD<7:0> (PORTC). PAD<15:8> (PORTB) will be set to 0xFF. Reading a configuration location between 0xFE08 and 0xFE0F will place the high byte of the configuration word into PAD<7:0> (PORTC). PAD<15:8> (PORTB) will be set to 0xFF.

### TABLE 4-1: CONFIGURATION BIT PROGRAMMING LOCATIONS

<table>
<thead>
<tr>
<th>Bit</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>FOSC0</td>
<td>0xFE00</td>
</tr>
<tr>
<td>FOSC1</td>
<td>0xFE01</td>
</tr>
<tr>
<td>WDTPS0</td>
<td>0xFE02</td>
</tr>
<tr>
<td>WDTPS1</td>
<td>0xFE03</td>
</tr>
<tr>
<td>PM0</td>
<td>0xFE04</td>
</tr>
<tr>
<td>PM1</td>
<td>0xFE06</td>
</tr>
<tr>
<td>PM2†</td>
<td>0xFE0F</td>
</tr>
</tbody>
</table>

†This location does not exist on the PIC17C42.

### TABLE 4-2: READ MAPPING OF CONFIGURATION BITS

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>—</td>
<td>—</td>
<td>PM1</td>
<td>—</td>
<td>PM0</td>
<td>WDTPS1</td>
<td>WDTPS0</td>
<td>FOSC1</td>
</tr>
</tbody>
</table>

— Unused

**PM<2:0>**, Processor Mode Select bits
111 = Microprocessor mode
110 = Microcontroller mode
101 = Extended Microcontroller mode
000 = Code protected microcontroller mode

**WDTPS<1:0>**, WDT Prescaler Select bits.
11 = WDT enabled, postscaler = 0
10 = WDT enabled, postscaler = 256
01 = WDT enabled, postscaler = 64
00 = WDT disabled, 16-bit overflow timer

**FOSC<1:0>**, Oscillator Select bits
11 = EC oscillator
10 = XT oscillator
01 = RC oscillator
00 = LF oscillator

* This bit does not exist on PIC17C42.
4.2 Embedding Configuration Word Information in the Hex File

To allow portability of code, a PIC17C4X programmer is required to read the configuration word locations from the hex file when loading the hex file. If configuration word information was not present in the hex file then a simple warning message may be issued. Similarly, while saving a hex file, all configuration word information must be included. An option to not include the configuration word information may be provided. When embedding configuration word information in the hex file, it should be to address FE00h.

Microchip Technology Inc. feels strongly that this feature is important for the benefit of the end customer.

### TABLE 4-3: CONFIGURATION WORD

#### PIC17C42
To code protect:
- Protect all memory  xxxxxxxxxxxxx

<table>
<thead>
<tr>
<th>Program Memory Segment</th>
<th>R/W in Protected Mode</th>
<th>R/W in Unprotected Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration Word (0xFE00)</td>
<td>Read Scrambled, Write Enabled</td>
<td>Read Unscrambled, Write Enabled</td>
</tr>
<tr>
<td>All memory</td>
<td>Read Scrambled, Write Disabled*</td>
<td>Read Unscrambled, Write Enabled</td>
</tr>
</tbody>
</table>

#### PIC17C42A
To code protect:
- Protect all memory  0xxxxxxxx0xxxxx

<table>
<thead>
<tr>
<th>Program Memory Segment</th>
<th>R/W in Protected Mode</th>
<th>R/W in Unprotected Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration Word (0xFE00)</td>
<td>Read Scrambled, Write Enabled</td>
<td>Read Unscrambled, Write Enabled</td>
</tr>
<tr>
<td>All memory</td>
<td>Read Scrambled, Write Disabled*</td>
<td>Read Unscrambled, Write Enabled</td>
</tr>
</tbody>
</table>

#### PIC17CR42
To code protect:
- Protect all memory  0xxxxxxxx0xxxxx

<table>
<thead>
<tr>
<th>Program Memory Segment</th>
<th>R/W in Protected Mode</th>
<th>R/W in Unprotected Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration Word (0xFE00)</td>
<td>Read Scrambled, Write Enabled</td>
<td>Read Unscrambled, Write Enabled</td>
</tr>
<tr>
<td>All memory</td>
<td>Read Scrambled, Write Disabled*</td>
<td>Read Unscrambled, Write Enabled</td>
</tr>
</tbody>
</table>

#### PIC17C43
To code protect:
- Protect all memory  0xxxxxxxx0xxxxx

<table>
<thead>
<tr>
<th>Program Memory Segment</th>
<th>R/W in Protected Mode</th>
<th>R/W in Unprotected Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration Word (0xFE00)</td>
<td>Read Scrambled, Write Enabled</td>
<td>Read Unscrambled, Write Enabled</td>
</tr>
<tr>
<td>All memory</td>
<td>Read Scrambled, Write Disabled*</td>
<td>Read Unscrambled, Write Enabled</td>
</tr>
</tbody>
</table>

#### PIC17CR43
To code protect:
- Protect all memory  0xxxxxxxx0xxxxx

<table>
<thead>
<tr>
<th>Program Memory Segment</th>
<th>R/W in Protected Mode</th>
<th>R/W in Unprotected Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration Word (0xFE00)</td>
<td>Read Scrambled, Write Enabled</td>
<td>Read Unscrambled, Write Enabled</td>
</tr>
<tr>
<td>All memory</td>
<td>Read Scrambled, Write Disabled*</td>
<td>Read Unscrambled, Write Enabled</td>
</tr>
</tbody>
</table>

#### PIC17C44
To code protect:
- Protect all memory  0xxxxxxxx0xxxxx

<table>
<thead>
<tr>
<th>Program Memory Segment</th>
<th>R/W in Protected Mode</th>
<th>R/W in Unprotected Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration Word (0xFE00)</td>
<td>Read Scrambled, Write Enabled</td>
<td>Read Unscrambled, Write Enabled</td>
</tr>
<tr>
<td>All memory</td>
<td>Read Scrambled, Write Disabled*</td>
<td>Read Unscrambled, Write Enabled</td>
</tr>
</tbody>
</table>

Legend: X = Don’t care

*Write to on-chip EPROM memory is disabled. The only way these locations can be programmed is if a TABLWT instruction is issued from an “on-chip” program memory space to program an on-chip memory location.*
4.3 CHECKSUM COMPUTATION

The checksum is calculated by summing the following:
• The contents of all program memory locations
• The configuration word, appropriately masked
• Masked ID locations (when applicable)

The least significant 16 bits of this sum is the checksum.

The following table describes how to calculate the checksum for each device. Note that the checksum calculation differs depending on the code protect setting. Since the program memory locations read out differently depending on the code protect setting, the table describes how to manipulate the actual program memory values to simulate the values that would be read from a protected device. When calculating a checksum by reading a device, the entire program memory can simply be read and summed. The configuration word and ID locations can always be read.

Note that some older devices have an additional value added in the checksum. This is to maintain compatibility with older device programmer checksums.

<table>
<thead>
<tr>
<th>Device</th>
<th>Code Protect</th>
<th>Checksum*</th>
<th>Blank Value</th>
<th>0xC0DE at 0 and max address</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC17C42</td>
<td>MP mode</td>
<td>SUM[0x000:0x7FF] + CFGW &amp; 0x005F + 0xFFA0</td>
<td>0xFF7F</td>
<td>0x79BD</td>
</tr>
<tr>
<td></td>
<td>MC mode</td>
<td>SUM[0x000:0x7FF] + CFGW &amp; 0x005F + 0xFFA0</td>
<td>0xFF7F</td>
<td>0x79AD</td>
</tr>
<tr>
<td></td>
<td>EMC mode</td>
<td>SUM[0x000:0x7FF] + CFGW &amp; 0x005F + 0xFFA0</td>
<td>0xFF7F</td>
<td>0x797D</td>
</tr>
<tr>
<td></td>
<td>PMC mode</td>
<td>SUM_XNOR8[0x000:0x7FF] + CFGW &amp; 0x005F + 0xFFA0</td>
<td>0xFF7F</td>
<td>0xBB73</td>
</tr>
<tr>
<td>PIC17C42A</td>
<td>MP mode</td>
<td>SUM[0x000:0x7FF] + CFGW &amp; 0x015F</td>
<td>0xF9F5</td>
<td>0x7B1D</td>
</tr>
<tr>
<td></td>
<td>MC mode</td>
<td>SUM[0x000:0x7FF] + CFGW &amp; 0x015F</td>
<td>0xF9F4</td>
<td>0x7B0D</td>
</tr>
<tr>
<td></td>
<td>EMC mode</td>
<td>SUM[0x000:0x7FF] + CFGW &amp; 0x015F</td>
<td>0xF9F1</td>
<td>0x7ADD</td>
</tr>
<tr>
<td></td>
<td>PMC mode</td>
<td>SUM_XNOR8[0x000:0x7FF] + CFGW &amp; 0x015F</td>
<td>0xF8F0</td>
<td>0xBBD3</td>
</tr>
<tr>
<td>PIC17CR42</td>
<td>MP mode</td>
<td>SUM[0x000:0x7FF] + CFGW &amp; 0x015F</td>
<td>0xF9F5</td>
<td>0x7B1D</td>
</tr>
<tr>
<td></td>
<td>MC mode</td>
<td>SUM[0x000:0x7FF] + CFGW &amp; 0x015F</td>
<td>0xF9F4</td>
<td>0x7B0D</td>
</tr>
<tr>
<td></td>
<td>EMC mode</td>
<td>SUM[0x000:0x7FF] + CFGW &amp; 0x015F</td>
<td>0xF9F1</td>
<td>0x7ADD</td>
</tr>
<tr>
<td></td>
<td>PMC mode</td>
<td>SUM_XNOR8[0x000:0x7FF] + CFGW &amp; 0x015F</td>
<td>0xF8F0</td>
<td>0xBBD3</td>
</tr>
<tr>
<td>PIC17C43</td>
<td>MP mode</td>
<td>SUM[0x000:0xFFF] + CFGW &amp; 0x015F</td>
<td>0xF1F5</td>
<td>0x731D</td>
</tr>
<tr>
<td></td>
<td>MC mode</td>
<td>SUM[0x000:0xFFF] + CFGW &amp; 0x015F</td>
<td>0xF1F4</td>
<td>0x730D</td>
</tr>
<tr>
<td></td>
<td>EMC mode</td>
<td>SUM[0x000:0xFFF] + CFGW &amp; 0x015F</td>
<td>0xF1F1</td>
<td>0x72DD</td>
</tr>
<tr>
<td></td>
<td>PMC mode</td>
<td>SUM_XNOR8[0x000:0xFFF] + CFGW &amp; 0x015F</td>
<td>0xF0F0</td>
<td>0xB3D3</td>
</tr>
<tr>
<td>PIC17CR43</td>
<td>MP mode</td>
<td>SUM[0x000:0xFFF] + CFGW &amp; 0x015F</td>
<td>0xF1F5</td>
<td>0x731D</td>
</tr>
<tr>
<td></td>
<td>MC mode</td>
<td>SUM[0x000:0xFFF] + CFGW &amp; 0x015F</td>
<td>0xF1F4</td>
<td>0x730D</td>
</tr>
<tr>
<td></td>
<td>EMC mode</td>
<td>SUM[0x000:0xFFF] + CFGW &amp; 0x015F</td>
<td>0xF1F1</td>
<td>0x72DD</td>
</tr>
<tr>
<td></td>
<td>PMC mode</td>
<td>SUM_XNOR8[0x000:0xFFF] + CFGW &amp; 0x015F</td>
<td>0xF0F0</td>
<td>0xB3D3</td>
</tr>
<tr>
<td>PIC17C44</td>
<td>MP mode</td>
<td>SUM[0x000:0x1FFF] + CFGW &amp; 0x015F</td>
<td>0xE1F5</td>
<td>0x631D</td>
</tr>
<tr>
<td></td>
<td>MC mode</td>
<td>SUM[0x000:0x1FFF] + CFGW &amp; 0x015F</td>
<td>0xE1F4</td>
<td>0x630D</td>
</tr>
<tr>
<td></td>
<td>EMC mode</td>
<td>SUM[0x000:0x1FFF] + CFGW &amp; 0x015F</td>
<td>0xE1F1</td>
<td>0x62DD</td>
</tr>
<tr>
<td></td>
<td>PMC mode</td>
<td>SUM_XNOR8[0x000:0x1FFF] + CFGW &amp; 0x015F</td>
<td>0xE0F0</td>
<td>0xA3D3</td>
</tr>
</tbody>
</table>

Legend: CFGW = Configuration Word
SUM[a:b] = [Sum of locations a to b inclusive]
SUM_XNOR8(a:b) = [Sum of 8-bit wide XNOR copied into upper and lower byte, of locations a to b inclusive]
*Checksum = [Sum of all the individual expressions] MODULO [0xFFFF]
+ = Addition
& = Bitwise AND
### 5.0 AC/DC CHARACTERISTICS

#### TIMING REQUIREMENTS FOR PROGRAM/VERIFY TEST MODE

**Standard Operating Conditions**

- Operating Temperature: +10°C ≤ TA ≤ +70°C, unless otherwise stated, (25°C is recommended)
- Operating Voltage: 4.5V ≤ VDD ≤ 5.25V, unless otherwise stated.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PD1</td>
<td>VDDP</td>
<td>Supply voltage during programming</td>
<td>4.75</td>
<td>5.0</td>
<td>5.25</td>
<td>V</td>
<td></td>
</tr>
<tr>
<td>PD2</td>
<td>IDD</td>
<td>Supply current during programming</td>
<td></td>
<td></td>
<td>50</td>
<td>mA</td>
<td>Freq = 10MHz, VDD = 5.5V Note 3</td>
</tr>
<tr>
<td>PD3</td>
<td>VDDV</td>
<td>Supply voltage during verify programming</td>
<td>VDD min.</td>
<td></td>
<td>VDD max.</td>
<td>V</td>
<td>Note 2</td>
</tr>
<tr>
<td>PD4</td>
<td>VPP</td>
<td>Voltage on VPP/MCLR pin during programming</td>
<td>12.75</td>
<td></td>
<td>13.25</td>
<td>V</td>
<td>Note 1</td>
</tr>
<tr>
<td>PD6</td>
<td>IPP</td>
<td>Programming current on VPP/MCLR pin</td>
<td>25</td>
<td></td>
<td>50</td>
<td>mA</td>
<td>Note 3</td>
</tr>
<tr>
<td>P1</td>
<td>FOSC</td>
<td>Osc/clockin frequency during programming</td>
<td>4</td>
<td></td>
<td>10</td>
<td>MHz</td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td>TCY</td>
<td>Instruction cycle</td>
<td>1</td>
<td>0.4</td>
<td>μs</td>
<td></td>
<td>TCY = 4/FOSC</td>
</tr>
<tr>
<td>P3</td>
<td>TirV2sH</td>
<td>RA0, RA1, RA2, RA3, RA4 setup before TEST↑</td>
<td>1</td>
<td></td>
<td>μs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P4</td>
<td>TtsH2mch</td>
<td>TEST↑ to MCLR↑</td>
<td>1</td>
<td></td>
<td>μs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P5</td>
<td>TbcV2irH</td>
<td>RC&lt;7:0&gt;, RB&lt;7:0&gt; valid to RA1 or RA0↑:Address/Data input setup time</td>
<td>0</td>
<td></td>
<td>μs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P6</td>
<td>TirH2bcl</td>
<td>RA1 or RA0↑ to RB&lt;7:0&gt;, RC&lt;7:0&gt; invalid ; Address data hold time;</td>
<td>10 TCY</td>
<td></td>
<td>μs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P7</td>
<td>T0ckiL2bcriZ</td>
<td>RT↓ to RB&lt;7:0&gt;, RC&lt;7:0&gt; high impedance</td>
<td>8 TCY</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P8</td>
<td>T0ckiH2bcV</td>
<td>RA1↑ to data out valid</td>
<td>10 TCY</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P9</td>
<td>Tprog</td>
<td>Programming pulse width</td>
<td>10</td>
<td>100</td>
<td>1000</td>
<td>μs</td>
<td></td>
</tr>
<tr>
<td>P10</td>
<td>TirH2irL</td>
<td>RA0, RA1 high pulse width</td>
<td>10 TCY</td>
<td></td>
<td></td>
<td>μs</td>
<td></td>
</tr>
<tr>
<td>P11</td>
<td>TirL2irH</td>
<td>RA0, RA1 low pulse width</td>
<td>10 TCY</td>
<td></td>
<td></td>
<td>μs</td>
<td></td>
</tr>
<tr>
<td>P12</td>
<td>T0ckiV2inL</td>
<td>RA1↑ before INT↓ (to go from prog cycle to verify w/o increment)</td>
<td>0</td>
<td></td>
<td>μs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P13</td>
<td>TinL2rtl</td>
<td>RA1 valid after RA0 (to select increment or no increment going from program to verify cycle)</td>
<td>10 TCY</td>
<td></td>
<td>μs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P14</td>
<td>Tvpps</td>
<td>VPP setup time before RA0↑</td>
<td>100</td>
<td></td>
<td></td>
<td>μs</td>
<td>Note 1</td>
</tr>
<tr>
<td>P15</td>
<td>Tvpph</td>
<td>VPP hold time after INT↓</td>
<td>0</td>
<td></td>
<td></td>
<td>μs</td>
<td>Note 1</td>
</tr>
<tr>
<td>P16</td>
<td>TvdV2tsH</td>
<td>VDD stable to TEST↑</td>
<td>10</td>
<td></td>
<td></td>
<td>ms</td>
<td></td>
</tr>
<tr>
<td>P17</td>
<td>TrbV2mcH</td>
<td>RB input (E1h) valid to VPP/MCLR↑</td>
<td>0</td>
<td></td>
<td></td>
<td>μs</td>
<td></td>
</tr>
<tr>
<td>P18</td>
<td>TmcH2rbi</td>
<td>RB input (E1h) hold after VPP/MCLR↑</td>
<td>10 TCY</td>
<td></td>
<td></td>
<td>ns</td>
<td></td>
</tr>
<tr>
<td>P19</td>
<td>TvpL2vdL</td>
<td>VDD power down after VPP power down</td>
<td>10</td>
<td></td>
<td></td>
<td>ms</td>
<td></td>
</tr>
</tbody>
</table>

**Notes:**

1. VPP/MCLR pin must only be equal to or greater than VDD at times other than programming.
2. Program must be verified at the minimum and maximum VDD limits for the part.
3. These parameters are for design guidance only and are not tested nor characterized.
Programming Mode entry | Load address X | Verify location X | Verify location X + 1 | Program location X + 1 | Do not increment PC by raising RA1 before RA0↓ | Verify location X + 1
---|---|---|---|---|---|---

Note: RA2 = 0
RA3 = 0
RA4 = 1
FIGURE 5-2: PROGRAMMING AND VERIFY TIMINGS II

- **Test**
- **V<sub>pp</sub>/MCLR**
- **RA1 = 0**
- **RA0**
- **RA2**
- **RA3 = 0**
- **RA4 = 1**
- **E<sub>H</sub>**
- **ADDR<sub>HI</sub>**
- **ADDR<sub>LO</sub>**
- **DATA<sub>HI</sub>**
- **DATA<sub>LO</sub>**
- **RC<sub>7:0</sub>**
- **Note: RA2 = 0, RA3 = 0, RA4 = 1**

- **t<sub>tvpH</sub>**
- **t<sub>tvpps</sub>**
- **t<sub>tprog</sub>**
- **t<sub>P9</sub>**
- **t<sub>P14</sub>**
- **Move to verify cycle**
- **Prevent increment of PC by raising RA1 before RA1**
- **Load address X**
- **Programming mode entry**
- **Program location X**
- **Verify location X**
FIGURE 5-3: PROGRAMMING AND VERIFY TIMINGS III

RA1 RA0 RB<7:0> RC<7:0>

Verify location X Verify location X +1
Pulse RA1 to increment address to X +2

INC PC


Verify location X +2
Program location X
Do not increment PC

IN IN IN IN

RA1 RA0 RB<7:0> RC<7:0>

Verify location X
Program location X
Raise RA1 after RA0

INC PC


Verify location X

Note: Device in PGM mode
Test = +5V
VMIC = Vpp
RA2 = 0
RA3 = 0
RA4 = 1

© 1996 Microchip Technology Inc.
FIGURE 5-4: POWER-UP/DOWN SEQUENCE FOR PROGRAMMING
Note the following details of the code protection feature on PICmicro® MCUs.

- The PICmicro family meets the specifications contained in the Microchip Data Sheet.
- Microchip believes that its family of PICmicro microcontrollers is one of the most secure products of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the PICmicro microcontroller in a manner outside the operating specifications contained in the data sheet. The person doing so may be engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable".
- Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our product.

If you have any further questions about this matter, please contact the local sales office nearest to you.

Information contained in this publication regarding device applications and the like is intended through suggestion only and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. No representation or warranty is given and no liability is assumed by Microchip Technology Incorporated with respect to the accuracy or use of such information, or infringement of patents or other intellectual property rights arising from such use or otherwise. Use of Microchip's products as critical components in life support systems is not authorized except with express written approval by Microchip. No licenses are conveyed, implicitly or otherwise, under any intellectual property rights.

Trademarks

The Microchip name and logo, the Microchip logo, FilterLab, KEELOQ, microID, MPLAB, PIC, PICmicro, PICMASTER, PICSTART, PRO MATE, SEEVAL and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

dsPIC, ECONOMONITOR, FanSense, FlexROM, fuzzyLAB, In-Circuit Serial Programming, ICSP, ICEPIC, microPort, Migratable Memory, MPASM, MPLIB, MPLINK, MPSIM, MXDEV, PICC, PICDEM, PICDEM.net, rfPIC, Select Mode and Total Endurance are trademarks of Microchip Technology Incorporated in the U.S.A.

Serialized Quick Turn Programming (SQTP) is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

© 2002, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

Printed on recycled paper.