

# The XPIC

Scott Masch

November 1, 2001

# 1 Introduction

The XPIC is a high speed microcontroller that is capable of running at 100MHz while executing one instruction per clock. It has 36 different instructions, and it uses a superset of Microchip Technology's 14-bit PIC instruction set which is used in their PIC16C6X series microcontrollers. Program memory is split between a 1K x 14 bit ROM (used for booting and test routines) and a 1K x 14 bit RAM (used for user programs and additional data storage). 96 bytes of memory are provided in the data address space for general purpose use. The processor itself is organized as a four stage pipelined for high performance. This design was fabricated in HP 0.5um CMOS (HP14TB) by the MOSIS fabrication service. We gratefully thank MOSIS for their support for this project.

The XPIC was thoroughly tested for functionality, performance, and power consumption. Functionality testing verified proper operation of all parts of the chip. Performance testing was done to find the maximum clock frequency of the chip at various voltages. The power consumption of the XPIC was also measured at various clock frequencies and voltages.

## 2 Design

The top level design of the XPIC is shown in Figure 1. This figure shows all of the major blocks of the chip.

### 2.1 Design Methods

The processor core of the XPIC was designed using Synopsys Protocol Compiler, which is a high level synthesis tool that uses a regular expression like input syntax and can generate VHDL, Verilog, or C output. Everything was converted into a netlist and technology mapped using Synopsys Design Compiler. Layout, routing, static timing analysis, and module building (for ROMs, RAMs, etc.) was done by Epoch from Cascade Design Automation (no longer in existence). Most of the simulation was performed using Model Sim. The boot code was tested using a serial EEPROM model generated with Synopsys MemPro. Finally, DRC and LVS checking was performed with Mentor Graphics CheckMate.

### 2.2 Processor

The XPIC is designed to run the Microchip 14-bit PIC instruction set with a few modifications. Four instructions have not been implemented; these are SLEEP (enter sleep mode), CLRWDT (clear watchdog timer), TRIS (write TRIS (data direction) register), and OPTION (write OPTION register). SLEEP and CLRWDT are not implemented because hardware support was never added, and TRIS and OPTION can already be implemented using `MOVWF loc`, where `loc` is the TRIS or OPTION register.

Three instructions have been added; these are TBLRD (table read), TBLRDT (table read using WREG), and TBLWT (table write). All of these instructions are used for reading and writing the program memory.

It is designed to run most instructions at the rate of one instruction per clock.

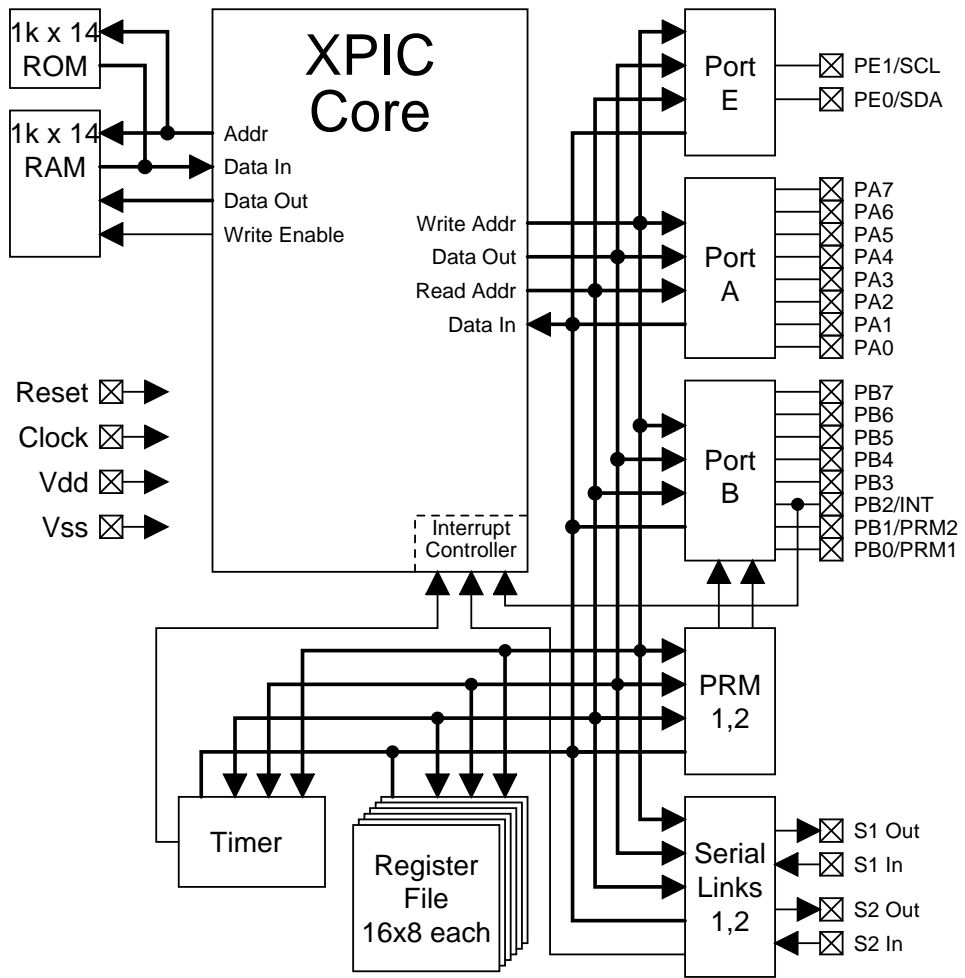


Figure 1: Top level design of the XPIC.

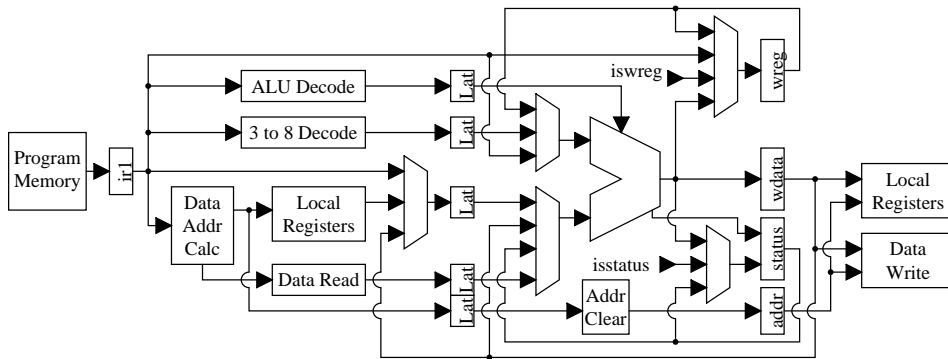


Figure 2: The XPIC datapath

### 2.2.1 Pipelining

In order to execute code at the rate of one instruction per clock, the XPIC uses a four stage pipeline. The structure of this pipeline is shown in Figure 2. The four stages are fetch, read/decode, execute, and write.

**Fetch** The fetch stage reads data from the program memory and latches it for the next stage.

**Read/Decode** The data needed for the instruction fetched is read from memory and latched. This is done regardless of whether the instruction needs the data or not, or if the instruction will even be executed. The data address is normally the low 7 bits of the instruction, however, if the address is zero (INDF), the actual address is the value contained in the FSR (File Select) register. For the data memory read, the test for a zero address is only done on the upper 4 bits of the address to allow more time for the data memory read. If this read is done unnecessarily because of a bypass or because data is not needed, the data will simply be ignored in the next stage.

A 3-to-8 decoder is used to create a 1-of-8 value for use in the bit operations (BSF, BCF, BTFSS, and BTFSC). The ALU operation bits are also calculated and latched in this stage.

**Execute** The execute stage uses a single-cycle ALU to compute the desired output value. The WREG (working register) value is always read and written to in this stage. The STATUS register is a special register that is also read and written to in this stage because it changes in (and is often read after) many arithmetic and data movement operations. The PCL register is written to in this stage to minimize delay in the jump that takes place because of the change of the program counter. The FSR register is also written here so that it is available for later instructions to use it as an indirect address.

This stage also selects if a write will take place during the write stage. If the current instruction is not being executed, or the current instruction does not write to memory, the address value for the write is converted to a location that does not write during the write stage. The two locations that are used are 0x03 (status register, written during the execute stage) and 0x07 (unused).

**Write** This final stage writes data back to memory. All locations are written in this stage except for INDF (not a real register), PCL (program counter low, written during execute), STATUS (written during execute), and FSR (also written during execute). As mentioned before, if a write is not needed, the write address is set to a location that is not written in this stage. This was done to reduce the number of bits that have to be tested to determine if a write should take place to a particular location and was found to reduce the area requirements of the chip compared to having a separate write flag.

### 2.2.2 Hazards

The XPIC core has several bypasses built in. There are two for read after write operations: one for a read immediately after a write and one for a read two cycles after a write. The latter is actually not necessary at lower speeds because the write to data memory can complete in time for the read to get the new data. However, above about 70 MHz (estimated from the timing data) this path is not fast enough so the internal bypass is needed. There are also hazards associated with some of the special function registers, these are discussed in the sections on these modules.

### 2.2.3 Interrupts

The XPIC has three interrupt sources: external edge, timer overflow, and serial link event. When any of these events occur, the corresponding bit in the INT register is set. If the enable bit for that event is set and interrupts are enabled in the STATUS register, an interrupt occurs. Interrupts take 3 clock cycles between when the first instruction is flushed and when the first interrupt routine instruction is executed. During these three cycles, the WREG and STATUS registers are backed up to shadow registers and the address of the first executable instruction during these three cycles is saved. This is needed in case a multicycle instruction (such as GOTO) is being executed, as we may not be executing the instruction currently in the execute phase. This means the interrupt and the multicycle instruction can share flush cycles and provide a slight performance improvement.


Interrupt routines are terminated by a RETFIE instruction. This instruction restores the STATUS, WREG, and program counter registers to their original states.

### 2.2.4 Special Function Registers

The special function registers (SFRs) and their locations are summarized in Table 1. These registers are used for controlling various parts of the chip. Details on SFRs specific to the processor core are as follows.

**INDF/FSR** The INDF and FSR registers are used for indirect addressing. Reading INDF reads the address referenced by FSR, and writing INDF writes the address referenced by FSR. FSR can be read or written to like any other register (it can even be read or written through INDF), although the topmost bit is fixed to one. When FSR is zero (the INDF address), reading INDF returns all zeros, and writing INDF effectively performs no operation (although STATUS bits may still be affected). There is one problem with the implementation of this on the XPIC: writing FSR does not take immediate effect (like it should). Instead, it takes one extra instruction cycle, meaning that at least one instruction must exist between a write of FSR and a read of INDF. (This is a well known bug, this was done because of chip area limitations and the desire to go fast.)

|      |                   |          |      |
|------|-------------------|----------|------|
| 0x00 | INDF <sup>1</sup> | PRM1H    | 0x10 |
| 0x01 | TMR               | PRM1L    | 0x11 |
| 0x02 | PCL               | PRM2H    | 0x12 |
| 0x03 | STATUS            | PRM2L    | 0x13 |
| 0x04 | FSR               | PORTB    | 0x14 |
| 0x05 | OPTION            | TRISB    | 0x15 |
| 0x06 | INT               | OUTB     | 0x16 |
| 0x07 |                   | PRMSPEED | 0x17 |
| 0x08 |                   | RXBUF    | 0x18 |
| 0x09 | TBLATH            | TXBUF    | 0x19 |
| 0x0A | TBLPTL            | MSGLEN   | 0x1A |
| 0x0B | TBLPTH            |          | 0x1B |
| 0x0C | PORTA             | STATREG  | 0x1C |
| 0x0D | TRISA             |          | 0x1D |
| 0x0E | OUTA              | SCONTROL | 0x1E |
| 0x0F | PORTE             |          | 0x1F |

 Unimplemented location, read values and write effects are undefined.

Note 1: INDF is not a physical register.

Table 1: Summary of Special Function Registers in the XPIC.

**PCL** Program Counter Low (PCL) contains the lower 8 bits of the 11 bit program counter. When read, it reads the address of the current instruction plus one. Writing this register causes a jump to STATUS[7:5]:PCL[7:0]. This can be used for indirect jumps. Since any instruction that writes PCL causes a change in the Program Counter, the instruction takes three cycles to execute and flush the pipeline.

**STATUS** The Status register holds the ALU status bits, the upper bits to write to the Program Counter during a write to PCL (discussed in section 2.2.4), and the Global Interrupt Enable bit. The bits of this register are



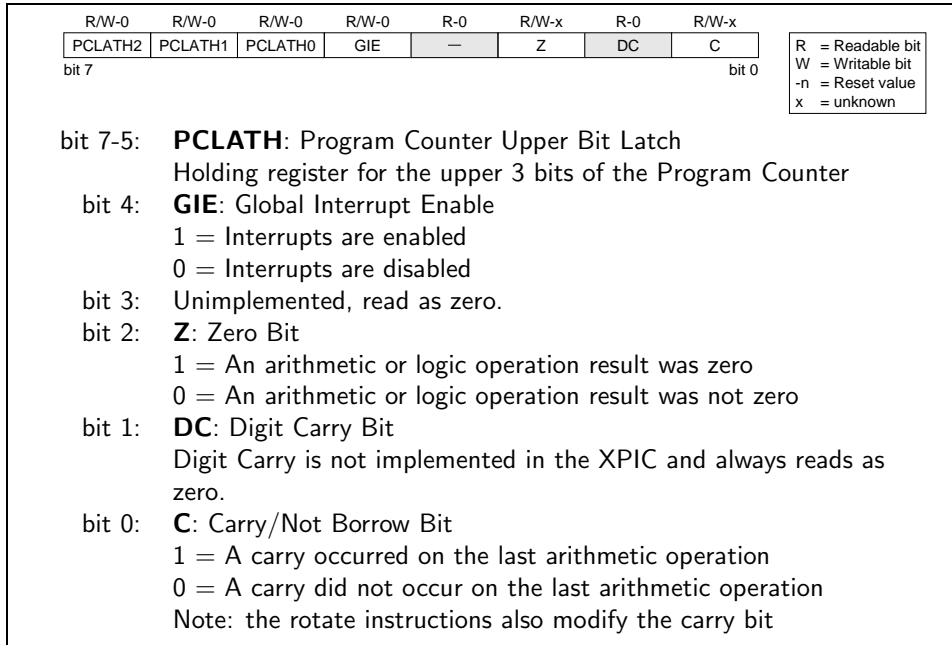


Figure 3: Status Register

shown in Figure 3. The Global Interrupt Enable bit enables and disables all interrupts. It is automatically cleared when an interrupt occurs and is restored to its previous value when the RETFIE instruction is executed. The Carry and Zero bits are updated by many arithmetic and logic operations. Writing to either of these two bits with any instruction that otherwise affects either of these bits will have the write disabled to these bits.

**OPTION** The OPTION register contains several bits controlling various resources and is shown in Figure 4. The Interrupt Edge Select bit is used to select which edge will be detected from the INT pin (Port B, bit 2). The T0EN and TPS bits are discussed in section 2.4 (Timer) and the PRM1EN and PRM2EN bits are discussed in section 2.3.3 (Pulse Rate Modulators).

**INT** The INT register contains both the interrupt flags and interrupt enable bits. This is shown in Figure 5. Whenever an interrupt event occurs, the interrupt flag for that source is set, regardless of the state of the Global Interrupt Enable bit (in Status) or the enable bit for that source. An interrupt occurs whenever the Global Interrupt Enable bit is set, an interrupt

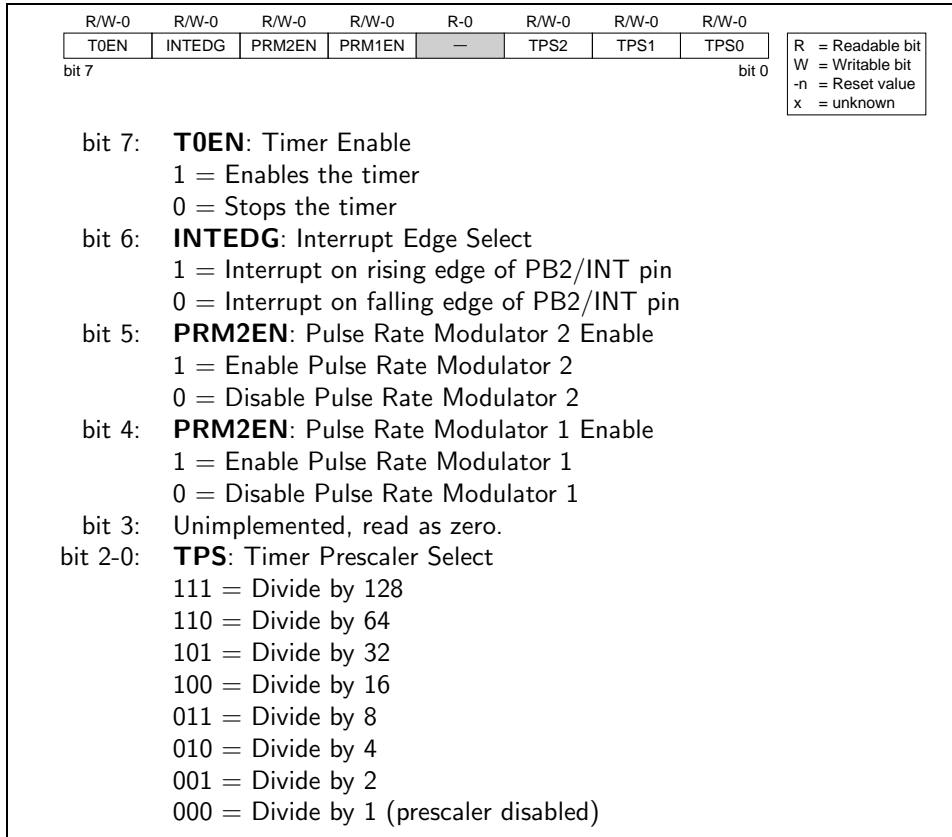


Figure 4: Option Register

flag is set, and the interrupt enable for that source is set.

**TBLATH** The TBLATH (Table Latch High) register stores the upper 6 bits of data for a TABLE operation. For a TBLRD or TBLRDT operation, the upper 6 bits of data read are loaded into this register. For a TBLWT operation, the upper 6 bits of data written are taken from this register. The lower 8 bits are read from/written to WREG. The upper two bits of this register are unimplemented and read as zero.

**TBLPTL and TBLPTH** TBLPTL (Table Pointer Low) provides the low 8 bits and TBLPTH (Table Pointer High) provides the high 3 bits used in TBLRD and TBLWT operations. For TBLRDT, only TBLPTH is

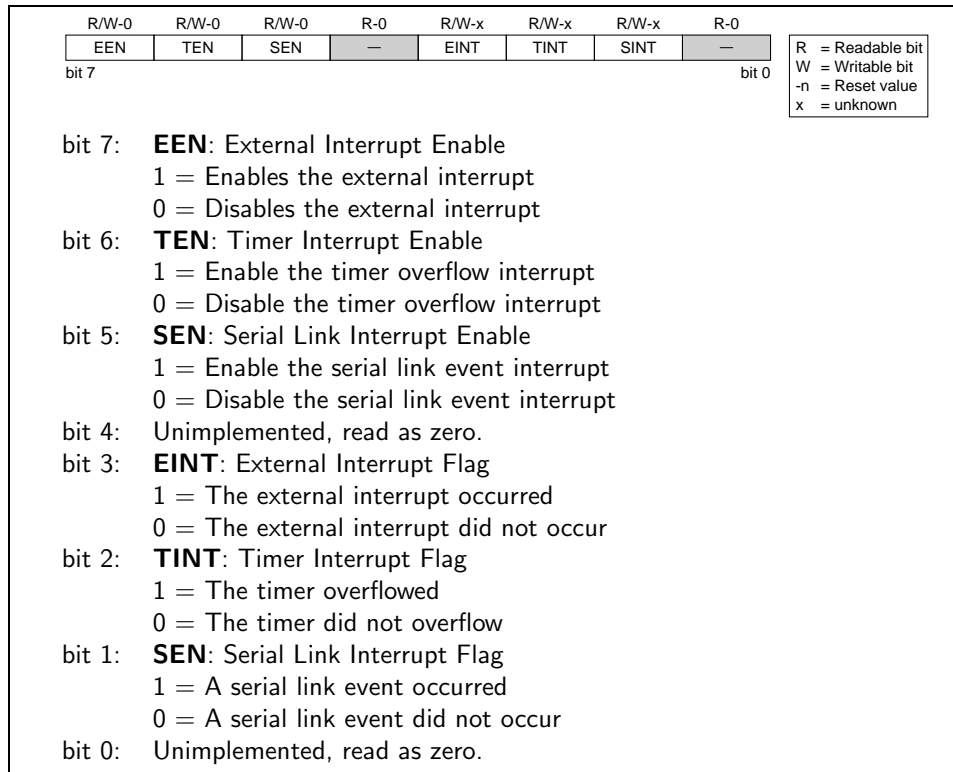


Figure 5: Interrupt Register

used; the low 8 bits come from WREG. The upper 5 bits of TBLPTH are unimplemented and read as zero.

## 2.3 I/O Ports

There are 18 general purpose I/O pins on the XPIC. Each pin can be used as either an input or an output.

### 2.3.1 Port A

There are three SFR addresses associated with Port A. The PORTA address writes to the output latch that sets the output port state and reads the actual port state. The OUTA address writes to the output latch like PORTA but reads the output latch value. This is useful in read-modify-write instructions

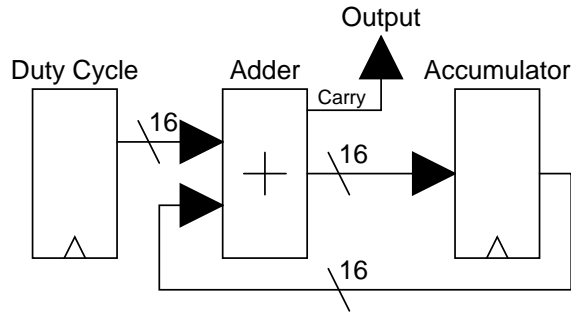


Figure 6: Simplified version of the pulse rate modulators used in the XPIC.

as this prevents bits from being erroneously changed while the port is an input. The TRISA address sets the direction for each pin of the port. One bits make pins inputs, while zeros make pins outputs.

### 2.3.2 Port B

There are three SFR addresses associated with Port B. These are PORTB, OUTB, and TRISB; and they all have the same functionality as the Port A addresses of the similar name.

### 2.3.3 Pulse Rate Modulators

There are two pulse rate modulators multiplexed with Port B. First of all, pulse rate modulation is similar to pulse width modulation in that there is a register that contains a desired duty cycle of an output pin. However, unlike pulse width modulation, the output pin transitions far more often (up to once per clock cycle) and often produces a cleaner output. A simplified version of the pulse rate modulators used in the XPIC is shown in Figure 6. The duty cycle is set with PRM1L and PRM1H for the first PRM and PRM2L and PRM2H for the second PRM. By design, the pulse rate modulators only run at half the core clock speed, and can be divided down to 1/510 of the core speed using an 8 bit LFSR. The speed is selected using the PRMSPEED register. The PRMs are enabled through the PRM1EN and PRM2EN bits in the OPTION register. Also, in order for the PRMs to do anything, the corresponding pins must be set as outputs and driven low.

### 2.3.4 Port E

Port E is a two pin port used primarily to read and write serial EEPROMs and to determine the boot mode. It has one register associated with it: PORTE. It has four active bits: two are equivalent to the PORT bits and two are equivalent to the TRIS bits.

## 2.4 Timer

The XPIC has a single 8-bit timer with a 7-bit prescaler. The timer is clocked by the instruction clock. The prescaler is capable of division by 1, 2, 4, 8, 16, 32, 64, and 128. The timer settings are controlled by the OPTION register, these are shown in Table 4. The timer is mapped to the data address space and can be read or written to at any time. When the timer overflows (from 0xFF to 0x00), the TINT bit in the INT register is set, causing an interrupt if interrupts are enabled.

## 2.5 Serial Links

The XPIC can communicate with other XPICs through one of two point-to-point serial interfaces on the chip. One serial link consists of two unidirectional lines which transmit at a rate 1/16 of core clock speed. The lines in either direction are independent, so it possible to transmit and receive simultaneously for full-duplex operation, as well as communicate on each serial link independently.

The protocol defined on these serial links is that a processor may transmit one packet at a time, each containing a two bit header and eight bits of data. Synchronization is done by the receiver upon seeing the first bit of the packet header. The receiver must reply to each packet received with a two bit acknowledgement pulse before the transmitter can send the next packet. At full speed, it takes approximately 14 serial clock cycles or 224 processor clock cycles to complete the transmission and acknowledgement of one packet. Thus, when the processor is running at 90 MHz, the speed of the serial links is 400 KB/s or 3.2 Mbps.

If a processor transmits a packet along a serial link, it expects to receive an acknowledgement within 16 serial clock cycles. If it does receive acknowledgement within this time, the transmission is considered a failure. At this

point, the byte is dropped, an error signal is sent to the processor, and the next packet is readied for transmission. Retransmission in the presence of errors must be done manually. Some instances in which a transmission error may occur is if there is noise on the line, causing the receiver to not interpret the incoming packet correctly, or if the receiver cannot accept the data because of a full buffer. The second condition will also cause an error signal to be sent to the processor on the receiver side.

Internally, each end of each line in the serial link is connected to a 4-byte FIFO. Each link can be programmed to send or receive messages of 1-4 bytes of data before signalling the processor. Such signalling can be done with an interrupt or by polling a bit in the serial link status register. Thus, it is possible to send or receive multi-byte packets with no processor intervention.

From a programmer's perspective, transmission is initiated by simply writing the data to transmit into a transmit buffer register. If the serial link is already busy, the data will wait in the transmit FIFO until the link is free. If the transmit FIFO is full, any data written to the transmit buffer register will be lost. On the receiving end, when a packet comes in, the programmer is responsible for reading the data from a receive buffer, and shifting out the data from the receive FIFO by writing a one to a bit in the serial control register. Failure to do so will cause an error when the receive FIFO overflows. The programmer is also responsible for clearing the serial status bits once a successful or unsuccessful transmission/reception has occurred.

## 2.6 Memory

The XPIC has  $1k \times 14$  of program ROM used for booting, testing, and various miscellaneous subroutines. When the XPIC starts up, it begins executing code from the beginning of ROM. This code resets some of the registers in data memory and reads Port E to determine the desired mode of operation. See Table 2 for the available settings. Two of the options are the ROM Self Test and the Sine Wave Test: these are discussed in Section 3. The third option is to boot from a serial EEPROM. In this mode, byte pairs are read from the beginning of the serial EEPROM and are loaded starting at the beginning of program RAM. This transfer is terminated and the loaded program is executed when the "end" bit in the byte pair is set.

| Port E2<br>SDA | Port E1<br>SCL | Run Mode                |
|----------------|----------------|-------------------------|
| 0              | 0              | ROM Self Test           |
| 0              | 1              | Boot from Serial EEPROM |
| 1              | 0              | Sine Wave Test          |
| 1              | 1              | Boot from Serial EEPROM |

Table 2: Port E pin states for boot mode selection.

The ROM also contains several “helper” subroutines for multiplication, serial EEPROM operations, and sine/cosine operations. There is also a 128-point quarter cosine table stored in program ROM to assist sine and cosine functions.

The XPIC has  $1k \times 14$  of program RAM used for executing user programs and storing general purpose data. Additionally, there are 96 bytes available in a register file for general purpose data storage located between 0x20 and 0x7F. Unlike the program RAM which can only be accessed using the table instructions, this data space can be directly accessed (or indirectly accessed using INDF/FSR) by many instructions.

## 3 Testing

### 3.1 Test Software

#### 3.1.1 ROM-Based Self Test

This is a series of programs intended to test most of the internal features of the chip. It includes tests for the execution unit, data memory, program memory, timer, interrupt, and serial links. For all of these tests, the Pulse Rate Modulators (PRMs) are both running in slow mode with PRM1 running with a 1/3 duty cycle and PRM2 running with a 2/7 duty cycle. A value indicating the current section of the code is written on Port B at various points in the code, while other current information is periodically written on Port A.

**Core Test** This is a long test that performs all sorts of basic arithmetic and logic operations, skips, direct and relative jumps, calls (using all 8 stack levels), direct and indirect memory accesses, and status register tests. These tests output an extensive amount of information on the I/O ports as failures here are likely to cause problems in later tests.

**Memory Tests** The program memory and data memory tests are very similar, so they will be discussed together. In the first test, pseudo-random data generated from a software based LFSR (9 bits wide for data memory, 17 bits wide for program memory) is written to all locations. Then this data is verified by re-running the LFSR routine and comparing the values. The second test is a variation of the March algorithm where alternating 1's and 0's are written to and later read from each location. (The only problem with this is that it is only performed at the word level, and not at the bit level.)

**Timer Test** This is a simple test where the timer is started, the program loops for a known amount of time, and then the value in the timer is verified with the correct value. This is done for the 1:1, 1:16, and 1:128 prescaler values.

**Interrupt Test** For the interrupt tests, a GOTO instruction is loaded at the interrupt vector (in program RAM) so that the interrupt routine in



ROM will be used. The interrupt routine sets a register in data memory to indicate which interrupts have occurred, clears the interrupt flags, and returns. First, the timer is turned on and the timer interrupt is enabled. The main loop waits for a length of time while counting the interrupts. This part of the test passes if the proper number of interrupts is generated before the time limit is exceeded. For the next test, timer interrupts are disabled and the external interrupt is enabled and high-to-low edge sensitivity is selected. A high-to-low edge is generated on the interrupt pin and the execution of an interrupt is verified. This is repeated for low-to-high edge sensitivity.

**Serial Link Test** All of these tests are first performed for transmission from serial port 1 to serial port 2, then for transmission from serial port 2 to serial port 1. This is done using the internal loopback. First, each byte is sent in sequence from 0 to 255, and reception of each byte is verified on the other side. Next, 5 bytes of data is sent without reading any of the received bytes, causing the receive FIFO (4 bytes deep) to overflow. This causes an error, which is verified before continuing. Last, the receive buffer is cleared, the loopback is turned off, and a byte is sent. This causes an error on the transmit side, which is verified.

### 3.1.2 RAM-Based Self Test

This is almost identical to the ROM-based self test, except that part of the self test code is present in program RAM. Actually, most of the self test code is still run from ROM, only the top level loop and the program memory test routine is in RAM. Unlike the ROM-based test, this version also toggles one of the serial EEPROM lines upon successful completion of a test loop as a simple pass/fail indication. The program memory test loop from the ROM was not used because this memory test is a destructive test, and would destroy the test program in RAM. The new program memory test only tests the upper 75% of the program memory leaving the test routine in the lower part of RAM alone.

### 3.1.3 I/O Port Test

This is a RAM-based program that performs a loopback test on the I/O ports. For this test, Port A and Port B must be externally connected together. First, Port A is set so all bits are outputs and then all combinations

of outputs are written onto the port. For each output combination, Port B is sampled to check for the correct value. Incorrect values are flagged and are later written out serially on Port E (the serial EEPROM port) for observation. This test is then repeated with the output latch on Port A set to zero and the data direction, TRIS, written. This causes the pins to only be driven low or be left floating. Since there are weak pullups on each pin (on the circuit board), each pin will be pulled high when the pin is an input. Like before, all possible combinations are written to Port A, and incorrect values are looked for on Port B. This procedure is repeated for sending from Port B to Port A.

### 3.1.4 Sine Wave Generation

This is a ROM-based program that generates sine waves on both of the PRM ports. The first PRM port has a single sine wave running at  $f_{clk}/32768$ . The second PRM port has two sine waves each of equal amplitude running at  $f_{clk}/32768$  and  $f_{clk}/163840$ . These sine waves are generated in software and use the quarter cosine table present in the ROM. Since the sine waves are generated with the PRM units (in fast mode), an external low pass filter is needed to be able to see the sine wave on an oscilloscope.

## 3.2 Test Setup

A printed circuit board (PCB) was made for testing the XPIC. This circuit board has a ZIF socket for the XPIC, extensive power supply decoupling, a socket for the serial EEPROM (for program code), connectors for all of the I/O and serial ports, a clock buffer, and a reset circuit. Figures 7 and 8 show these features of the PCB.

### 3.2.1 Test Socket

The XPIC was mounted using a ZIF socket (Aries Electronics #52-536-11). This socket was used because it was the only one we could find that could handle a 52 pin LCC device. Actually, this socket was made for PLCC chips and a block of foam (visible in Figures 10 and 11) had to be used to keep sufficient force on the pins.

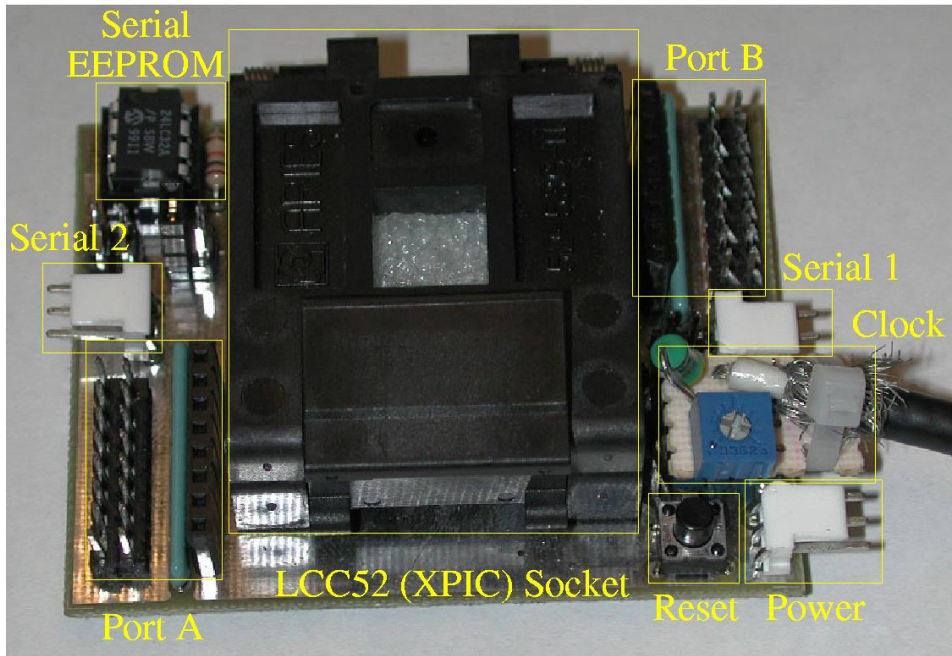


Figure 7: The XPIC test board.

### 3.2.2 Power Supply

A large number of decoupling capacitors were installed on the board to help keep the power supply stable at 100 MHz. These capacitors are chip capacitors mounted on the bottom of the board and are visible in Figure 8. The larger capacitors are  $0.1\mu\text{F}$  and are scattered throughout the board. The smaller capacitors (located directly underneath the XPIC) are both  $0.01\mu\text{F}$  and  $0.001\mu\text{F}$  and were added because of suspected power supply noise problems. The top layer of the board (shown in Figure 9 without any parts mounted) is primarily a ground plane, while the area around the XPIC is primarily a power plane; this was done to keep power supply impedance to a minimum. There is also a  $68\mu\text{F}$  capacitor mounted on the board to filter any low frequency noise, this is shown in Figure 8 and is located along the lower edge to the right of the reset circuit. Power enters the board through a three pin connector on one corner of the board.

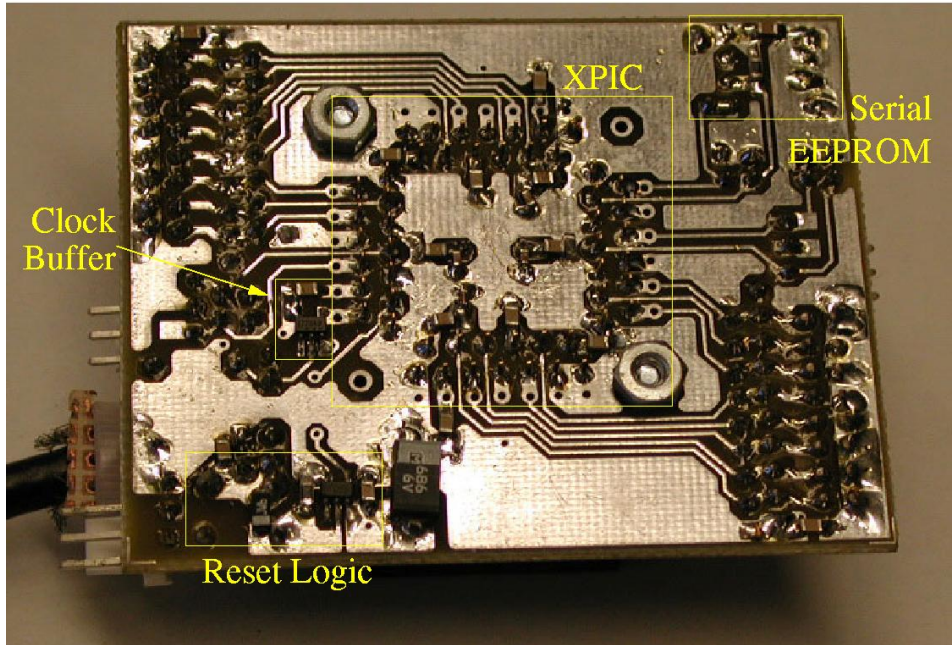


Figure 8: Bottom of the XPIC test board.

### 3.2.3 Reset

The reset circuit consists of a pushbutton, a resistor, a capacitor, and a Schmitt Trigger inverter. It will generate a 0.3 second reset pulse to the XPIC on power up and whenever the reset button is pressed. The reset circuit is shown in Figure 8. The 5-pin device on the right side of the reset circuit is a single Schmitt Trigger inverter (Fairchild #NC7S14).

### 3.2.4 Clock

The XPIC board has a 4-pin oscillator socket that can accept standard half-size oscillators. The clock line of this socket is connected through an inverter (Fairchild #NC7SZ04) to the clock input of the XPIC. This inverter is shown in Figure 8.

For most of the testing, a variable frequency clock source was used. For earlier parts of the testing, a high speed function generator was simply connected between the clock and ground lines of the oscillator socket. Later

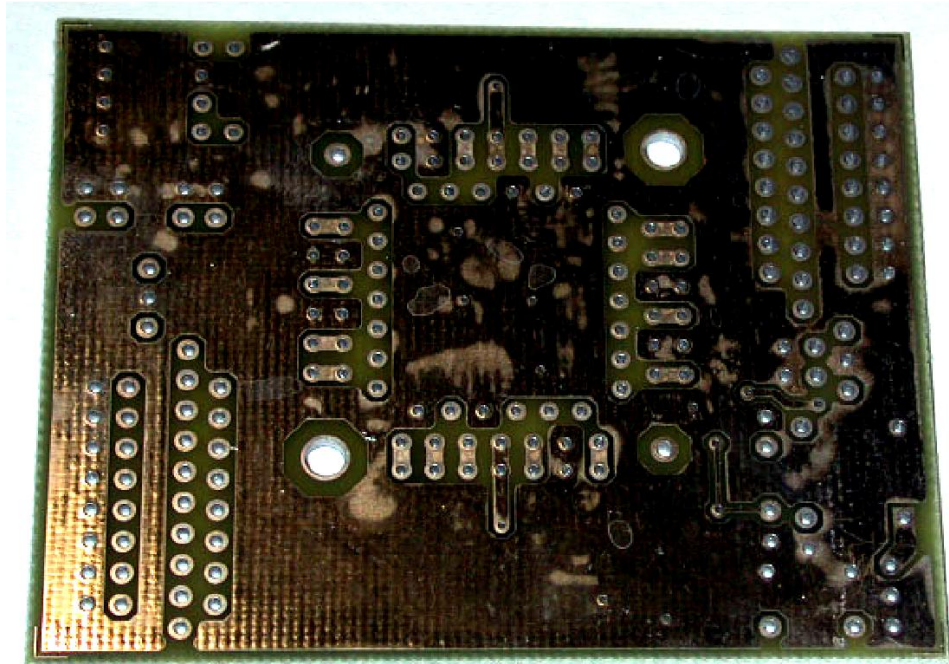


Figure 9: Top layer of the XPIC board without any components.

on, an RF generator was used, which required a DC block in order to be connected to the XPIC board. The DC block module is shown in Figure 7. It is constructed on Vector board, plugs directly into the 4 pin oscillator socket, and has an RG-58 cable soldered to it which connects to the RF generator. A  $47\Omega$  resistor terminates the cable while a  $0.01\mu\text{F}$  capacitor couples the clock signal to the XPIC board (both are surface mount parts on the bottom of the DC block board). A variable resistor and inductor sets the DC bias on the XPIC side of the DC block.

### 3.2.5 I/O Ports

Ports A and B (shown in Figure 7) can both be accessed via two connectors: a male header designed for connection to a logic analyzer and a female connector that can accept wires for easy connection. Each port pin has  $47\text{k}\Omega$  pullup resistor to hold the pin high when it is otherwise left floating.

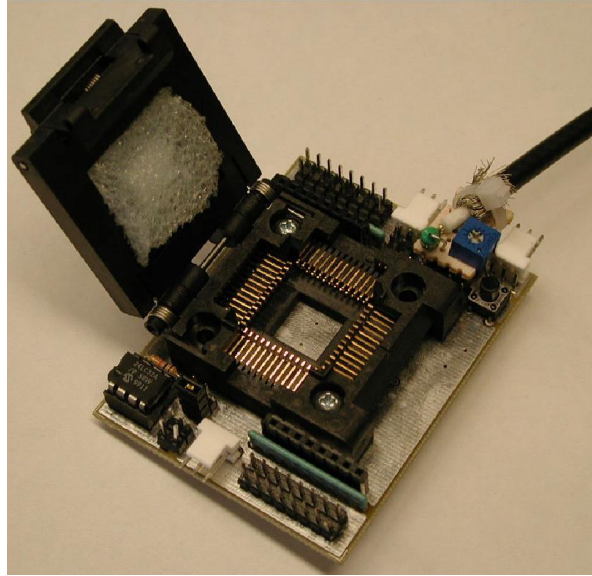


Figure 10: Another view of the test board showing the inside of the test socket.

### 3.2.6 Serial EEPROM/Boot Mode Selection

The serial EEPROM (shown in Figure 7) is mounted in a 8 pin DIP socket and can be removed for programming. These serial EEPROMs use a two wire I<sup>2</sup>C interface. The pullup resistors for I<sup>2</sup>C are mounted on the board. The boot mode selection jumpers are just below the serial EEPROM. In the figure, one of the pins has a jumper to select Sine Wave Generation.

### 3.2.7 Serial Ports

The three-pin connectors on each side of the board allow connection to the XPIC serial ports. The three pins are transmit, receive, and ground. The receive lines are pulled up through resistors on the board.

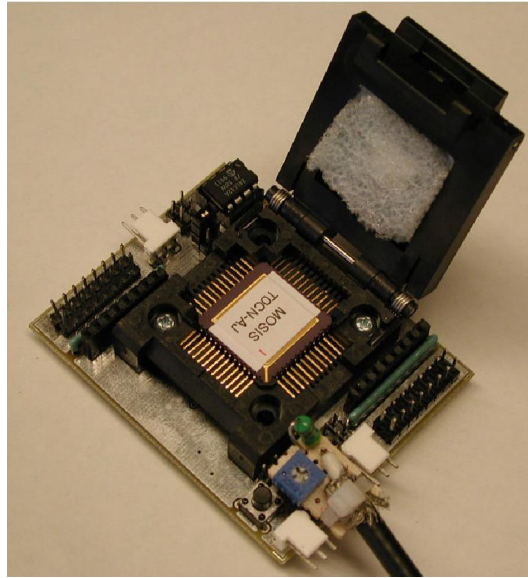


Figure 11: The test board with an XPIC chip in position in the socket.

### 3.3 Test Results

#### 3.3.1 Functionality

Most of the functionality testing was performed with the ROM-based self test. For these tests, a logic analyzer was connected to ports A and B. This provides enough information to verify proper operation of the entire self test and can also provide some debugging information if necessary. All of the chips were able to pass this test.

The I/O port test was also run. All ports were observed to be working correctly from port E data. Each loopback wire was then briefly removed: this causes a failure for that pin on each port, this was also observed on port E. For each port pair, a pull down resistor was briefly connected. This causes the drive low/high impedance test for both ports to fail, but it does not affect the fully driven test. These results are observed on port E. All chips were able to pass this test.

### 3.3.2 Performance

The maximum clock frequency for each of the 12 chips tested is shown in Figure 12. The maximum speed was found by running the RAM-based self test and adjusting the clock frequency up to the point where the device quit working properly, then adjusting the frequency back down until the device was stable. This frequency was considered to be the maximum frequency. This was repeated for each of the 12 chips and from 2.5 to 3.7 volts in 0.1 volt increments.

Three of the chips (numbers 6, 8, and 10) quit working before reaching the 2.5 volt lower limit. This is believed to be because the RAM sense amps ceased to work below some power supply voltage.

### 3.3.3 Power Usage

The power consumption of the XPIC was measured both during performance testing at maximum speed and later at a fixed frequency but variable voltage. During performance testing (described above in 3.3.2) each time the maximum frequency was found, the current at this frequency was also recorded. The results of this are shown in Figure 13. For the fixed frequency testing, a constant clock signal at 50 MHz was applied while the voltage was adjusted from the minimum operating voltage of the device to a little over 3.7 volts. The results of this test is shown in Figure 14.



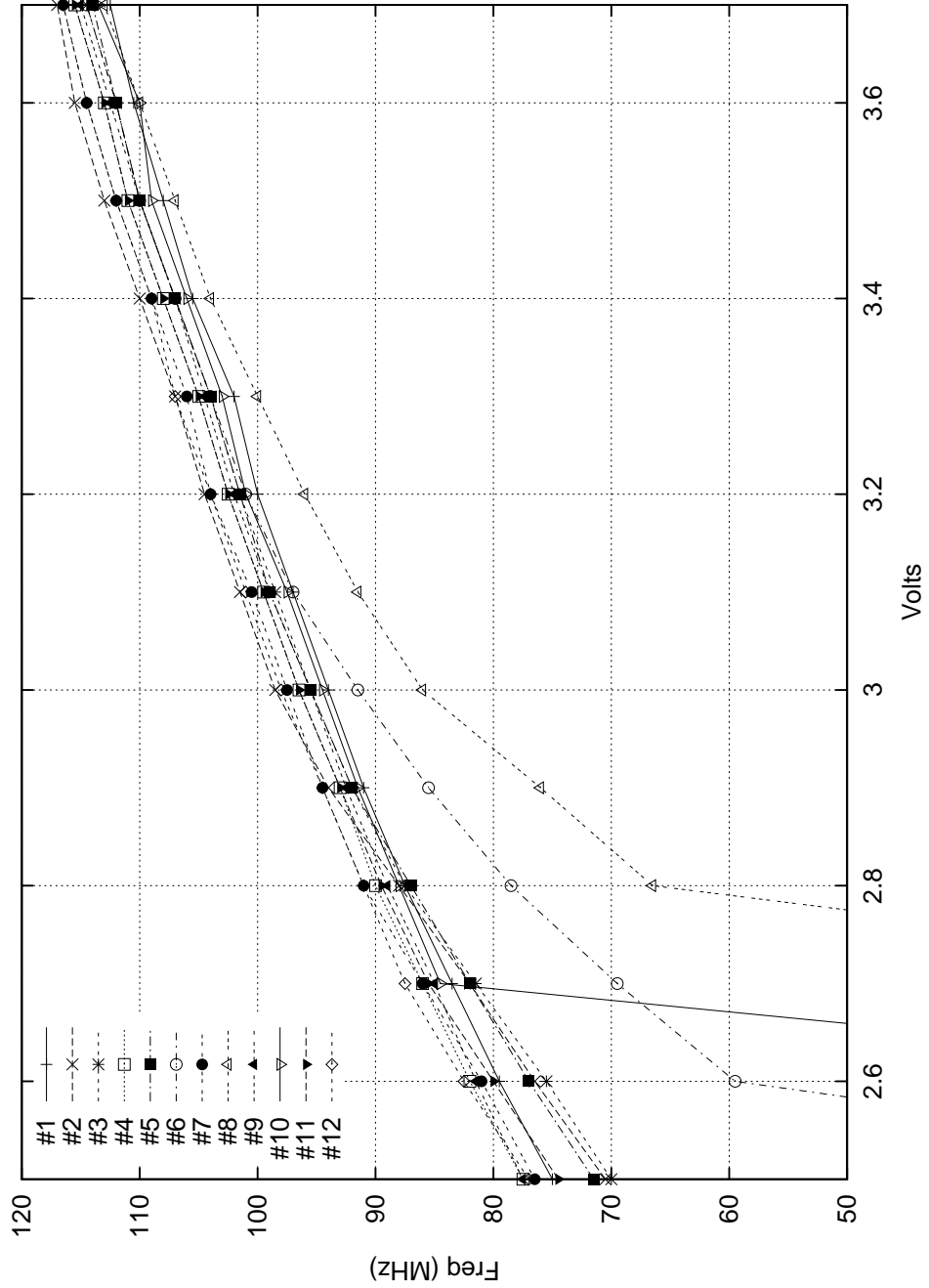


Figure 12: Maximum clock frequency at various voltages for each chip.

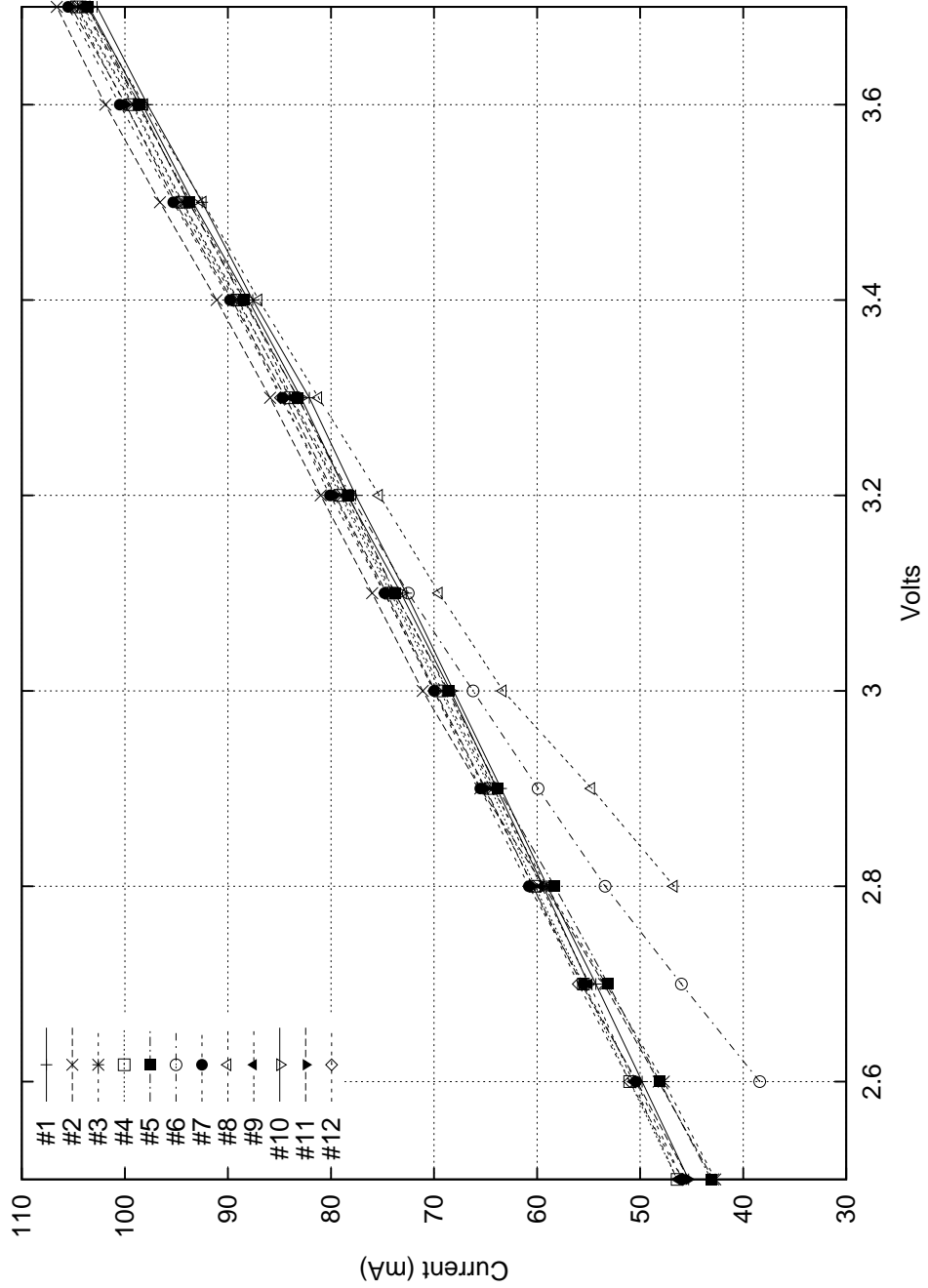


Figure 13: Current consumption at the maximum clock frequency.

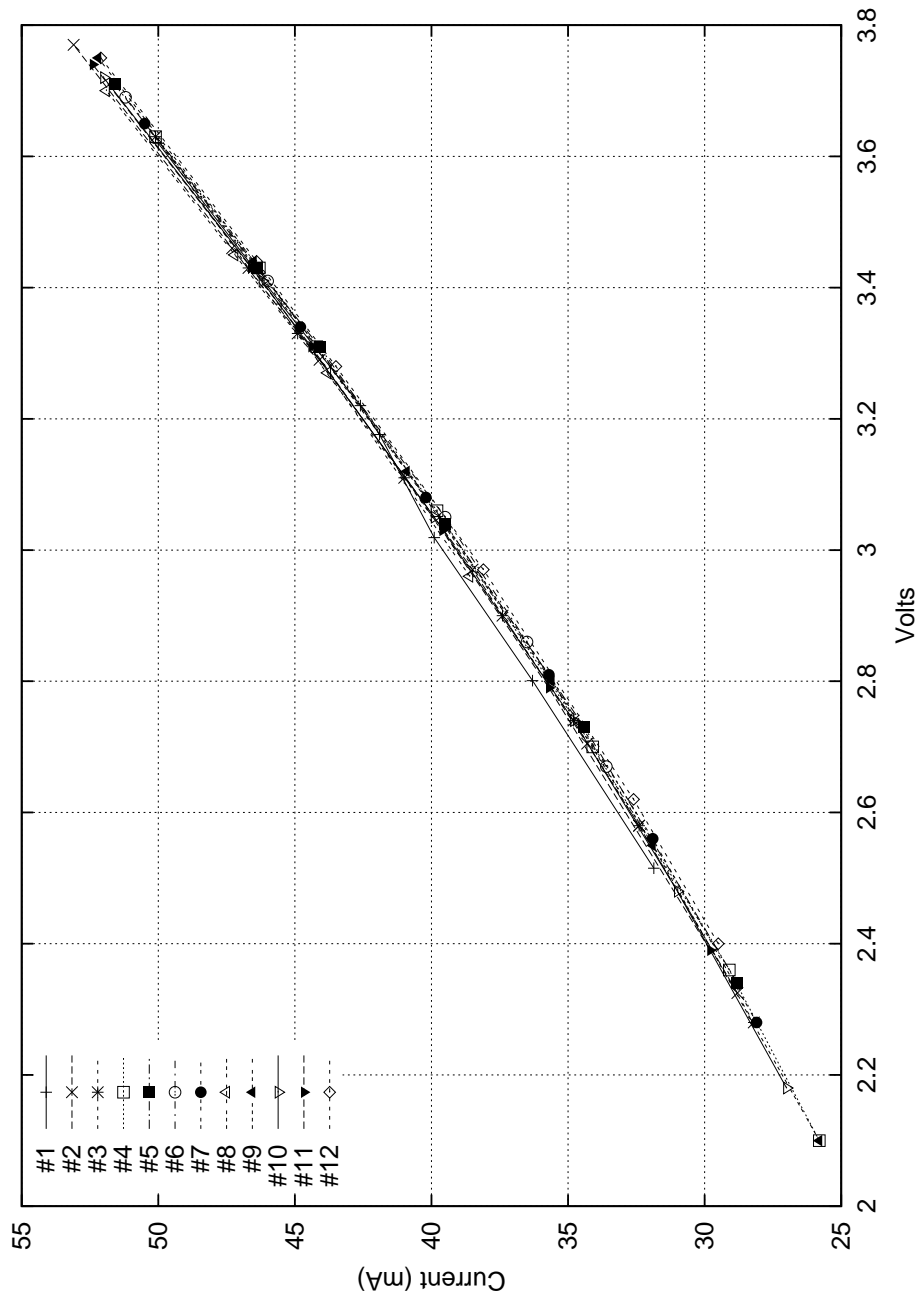


Figure 14: Current Consumption at 50 MHz from the minimum operating voltage to about 3.7 volts.

## 4 Conclusion

The XPIC seems to be functioning as it was designed. It runs slightly faster than we simulated. Late in the design phase, simulations showed the chip could run as fast as 92 MHz at 3.3 volts, while tests of the actual chip ran in excess of 100 MHz. This was probably because of process improvements since the timing models in Epoch were built. This could be compared to current professional designs – PIC offers a 40MHz part (0.25 IPC) and Scenix offers a similar part – running currently at 75MHz with an IPC of about 1.0. Our design uses more power than the Microchip design but offers 10X the performance, and beats the Scenix design in performance and power dissipation. We believe that this success is largely due to RT level design flow improvements which made possible rapid turnaround of architectural changes in the design. Back end flow used the Duet (Formerly Cascade Design Automation) Epoch tool which unfortunately is now defunct. This tool provided a very efficient (in design time) mechanism to handle low level changes. The design changed from 74 to 92 MHz in simulation by judicious floorplanning, buffering and low level placement changes.

Fairly extensive tests were run on the design at speeds requiring a custom circuit board assembly. Decoupling of this board was facilitated by numerous chip scale capacitors mounted at low inductance sites. In early stages of testing we had problems getting a clock into the chip without causing excessive power supply noise (about 330 mV P-P at 100 MHz). This problem did not appear when running a local crystal oscillator on the board. The problem was traced to line coupling and signal reflection at the termination. Accordingly, a D.C. blocking capacitor and resistive terminator were added, as well as a CMOS inverter acting as a clock buffer. The total noise was reduced to below 100 mV P-P over the test frequencies.

The core of the design was written using Synopsys Protocol Compiler, a high level tool, which allowed considerable freedom in automatic finite state machine synthesis. While this did take extra time in the beginning to learn the tool, it allowed for rapid design changes which were important to accommodate bypassing for the pipelined design. Further, the rapid turnaround made bug fixing easier since bugs could usually be confined to relatively small portions of the code.

This design contained a substantial amount of ROM to provide for boot and program loading (eprom was not an option in this process). The ROM also provided a convenient location for substantial built-in test code for

functional verification of the processor. This greatly simplified the creation of SHMOO plots since the test code provided a fast boot of the go-nogo functional test. This feature will doubtless be incorporated in future designs.