PIC Research: Shuriken Project

Connor Ledgerwood  
ECEN 4013  
Fall 2011

**Initial Research**

PIC Types

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Family | Word Size | Clock Speed | ROM | RAM | Price (Mouser) |
| PIC18F | 8-bit | 40 MHz | 128 KB | 4 KB | $4.88 |
| PIC24H | 16-bit | 60 MHz | 536 KB | 96 KB | $6.26 |
| PIC32 | 32-bit | 80 MHz | 512 KB | 128 KB | $7.95 |

\* All data represented as the maximum possible functionality for each PIC family

Package Models

|  |  |
| --- | --- |
| Model | Description |
| DIP | Too large, breadboard-oriented |
| SOIC | Solderable, elongated pin-out |
| TQFP | Solderable, square pin-out |
| QFN | Too small to prototype |
| PLCC | Too small to prototype |

Verdict

* PIC Model: 18F4520
  + Cheap
  + Solderable
  + Available in 44-pin TQFP
  + Fast enough to meet calculation needs
    - Accelerometer is the only resource-intensive unit
    - 8-bit operations are sufficient for most I/O
* Features
  + Three 16-bit timers
  + One 8-bit timer
  + 2 PWM units
  + 13 ADC channels
  + EUSART
  + SPI
  + I2C
  + MSSP
  + PSP

**Resource Allocation**

Modules Attached

* ADXL345 Digital Accelerometer
* XBEE Series-2 Pro Module
* PIC18F2580 Audio Slave
* 7-Segment display
* Ready LED
* Throw strength LEDs
* IR receivers
* IR transmitter
* Reload switch
* Programming header

Resources Needed

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Unit | Special Functionality | Ports | # Pins | Interrupts | Timers |
| Accelerometer | SPI | RC(5:2) | 4 | 0 | 0 |
| XBee | UART Rx/Tx | RC(7:6) | 2 | 1 | 0 |
| Audio Chip | Parallel protocol | RA(7:5), RE(1:0) | 5 | 0/1 | 0 |
| 7-Segment display | -- | RD(7:4) | 4 | 0 | 0 |
| Ready LED | -- | RA(4) | 1 | 1 | 1 |
| Throw Strength LEDs | -- | RA(3:0) | 4 | 1 | 0/1 |
| IR receivers | External interrupt pin | RB(2) | 1 | 1 | 1 |
| IR transmitter | Pulse Width Modulation | RB(3) | 1 | 0 | 1 |
| Reload switch | -- | RB(4) | 1 | 0 | 0 |
| Programming header | MCLR, PGD, PGC, PGM | RB(7:5), RE(3) | 4 | 0 | 0 |
| **Total:** | | | **27** | **5** | **4** |

The pins of the PIC18F4520 are not dynamic. Port allocation is given to external units based on two strategies.

1. Supported functionality of a pin (e.g. PWM output from RB3)
2. Locality with other pins used by a unit (e.g. 7-segment display uses consecutive ports)

**Audio Software**

Sound File Encoding

The main PIC sends a 3-bit encoded signal to the audio PIC to notify it of which song to play:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sound** |  | **Encoding** | | |
| *RA7* | *RA6* | *RA5* |
| No Sound | 0 | 0 | 0 |
| Proper Attack | 0 | 0 | 1 |
| Proper Reload | 0 | 1 | 0 |
| Faulty Action | 1 | 0 | 0 |
| Receive Damage | 1 | 1 | 1 |
| x | x | x | x |
| x | x | x | X |

Pseudocode (Main PIC)

/\*\*

\* RC = ReadControl = RE1

\* WC = WriteControl = RE0

\*

\* These pins are used in Master/Slave communication

\* between the main PIC and the audio chip

\*

\*\*/

Main\_PIC\_AUDIO(int audioRequest) {

// Clear former sound signal //

PORTA\_Audio |= ~0xE0;

// Output audio code //

PORTA\_Audio |= audioRequest;

// Read toggle pin's previous state //

int prevState = RC;

// Toggle pin to audioslave //

WC = ~WC;

// Wait for audioslave to toggle in acknowledgement //

while (RC == prevState)

nop;

}

**Motion Handling**

Brief Description

The ADXL345 is a digital accelerometer capable of both SPI and I2C serial protocol. As such, it was connected to pins on the PIC that could function with either protocol. For now we are choosing a 4-wire SPI interface for its seeming simplicity.

There are 3 axes with 2 polarities each for a total of 6 readable directions.

|  |  |
| --- | --- |
| **Methods Used** | |
| Core methods | Supplemental Methods |
| * readAxis()   + requests an axis reading   + stores it in an array   + flags when threshold is broken * monitorBreakthrough()   + reads several extra points when an axis falls below threshold   + concludes and resets the event when readings fall correctly below threshold * badData()   + reads several extra points when an axis falls below its threshold   + decides if the point that fell below threshold was just a bad reading   + sets up the algorithm to continue monitoring a breakthrough if it does decide so | * Init\_accelerometer()   + createAxis()   + setThreshold()   + initNumbers() * sendSPI()   + signals serial data with Chip Select bit   + specifies axis of interest in data packet * receiveSPI()   + waits for an acceleration reading to be returned * findMax()   + returns twice the average breakthrough value   + may need to be modified for more non-ideal situations * checkMaximums()   + analyzes the localMax values of all axe   + compares nonzero maximums with the expected response of a “good” action |

Pseudocode

struct axis {

int SPIcode;

int threshold;

int[] data;

int currentIndex;

int didBreak;

int numBreaks;

int sum;

int localMax;

}

void readAxis(struct\* axis) {

sendSPI(SPIcode);

data[index]=receiveSPI();

index = (index+1) % data.size;

if (data > threshold && didBreak == 0)

didBreak = 1;

}

void monitorBreakthrough(struct\* axis) {

if(didBreak)

sum += data[index];

numBreaks++;

else if(sum != 0)

int result = badData(axis);

if(result == 0)

localMax = findMax(axis);

sum = 0;

numBreaks = 0;

didBreak = 0;

}

int findMax(struct\* axis) {

int average = sum/numBreaks;

return 2\*average;

}

int badData(struct\* axis) {

int numExtraPoints = [[[???]]];

float ratio = [[[???]]]?

int count = 0;

int moreSum = 0;

int i;

for(i=0; i<extraPoints; i++)

readAxis(axis);

if(didBreak)

count++;

moreSum += data[index];

if(count/extraPoints > ratio)

sum += moreSum;

numBreaks += numExtraPoints;

return 1;

else

return 0;

}

void main\_accelerometer() {

int throwResult;

init\_accelerometer();

while(1) {

readAxis(X1);

monitorAxis(X1);

readAxis(X0);

monitorAxis(X0);

readAxis(Y1);

monitorAxis(Y1);

readAxis(Y0);

monitorAxis(Y0);

readAxis(Z1);

monitorAxis(Z1);

readAxis(Z0);

monitorAxis(Z0);

throwResult = checkMaximums();

if(throwResult == 1)

goodThrow();

else if (throwResult == -1)

badThrow();

else;

}

}

**Communication Protocols**

Devices

* XBee uses RS-232
* Accelerometer can use either SPI or I2C
* Audio slave uses a nonstandard parallel protocol

Since XBee must use the RS-232 Tx and Rx ports, the only real choice in choosing a protocol lies in the accelerometer. This is particularly important because most of the master PIC’s resources will be used in receiving data from each axis of the accelerometer. Below is a table comparing SPI and I2C.

|  |  |  |
| --- | --- | --- |
| Property | SPI | I2C |
| *Max Speed* | 5 MHz | 400 kHz |
| *Synchronous* | Yes | Yes |
| *Usefulness* | Ideal for imprecise oscillators | ACK system ensures transmission |
| *Activation* | CS (Chip Select) signals a slave to receive incoming data | Address field is included with each transmission |
| *External Parts* | None | Pull-up resistors |

Verdict: SPI

* Faster transmission speed
* Requires no extra parts
* We are using an internal oscillator
* The I2C address feature is redundant with only one device