# 8/8/2012

I ran through some example code for the A1Xplained kit. There's a demo program. I skimmed through it, but found that I was

unable to upload modified code to my kit. Atmel Studio 6 doesn't recognize it.

# 8/9/2012

I realized that the first reason it doesn't recognize it is that I don't have a debugger installed. I found a STK500v2 compatible AVR programmer by Olimex

that should be able to connect. I have also begun with the atmel Xmega training tutorials which I will do presently.

5:00 PM

I tried the training and it failed. I couldn't recognize the device. So I looked here:

http://www.avrfreaks.net/index.php?name=PNphpBB2&file=printview&t=109825&start=20 I found I need FLIP,

FINALLY got the thing working. Had about 6 or 7 drivers in a folder.. none of them were working. So I swithced the jumper cable

to JTAG 1 and 2 (top left and bottom left, facing it) and browsed for a driver. It worked. I'm happy. Still more to do though.

Following the Getting started guide.

Now I can load flip, select COM4, and load a hex file to load firmware into the board. Debugging doesn't work. But at least

I can upload code. I guess I can create an elaborate system of using LEDs to debug...

7:00 PM

Now I'm getting WinAVR to try that. Im reading this

http://fourwalledcubicle.com/blog/2010/02/mein-server/ to AVRISP-MKII make one

# 8/11/2012

I opened up AVR-MAX and started trying to port it. I quickly got confused as it includes a lot of windows headers. It doesn't seem to be for an AVR but it says it is.

Maybe it's simply a test program that uses the windows headers. Anyway, I found this thing called Xmega lab that has an adaption of the chess engine on board.

I am attempting to sift through it now to see if it takes a more embedded approach.

6:49 PM

I saw its approach was very integrated and thus it would be very difficult to test with. I added avrMicroMax as a codelite

project in the examples directory. I can build it and interact with it to test it. My goal is to figure out exactly how it works

by studying the code and stepping through it, and port the part that doesn't require a windows interface to the AVR project

while still exposing an interface to the main module. After that I'll examine a single test case in the debugger

and make sure it can be called through the main module in my AVR code by examining it on the simulator.

9:00 PM

Okay so I got the engine to work and I'm attempting to refactor it into something with a usable interface. currently everything

is in a main function and I'm studying the code to move pieces into the desired interface functions.

I found that the main loop is continually polling for xboard compatible engine commands. You can find more info

here: http://home.hccnet.nl/h.g.muller/engine-intf.html. What this means is that we need a new function that sends commands

directly to the engine instead of relying on standard IO. We need to send commands to this function from our other, higher level

interface functions.

11:29 PM

SUCCESS! It took a bit of hacking but I figured out how to make a basic control loop that moves player and then AI. I'm only taking into account player VS AI

game mode right now. There's a little bit of a problem with this interface. It is very hard coded and not object oriented. This means I'll need to keep a separate project

from the microcontroller code to test the interface. I'll need to really polish the interface using standard IO on my windows machine before I can send it to the microcontroller.

If it doesn't work flawlessly on the microcontroller, it will be INCREDIBLY difficult to debug. The code is just too confusing to not use a step debugger.

I'm still defining the interface as I go. For instance, I'll need to get a function to return whether or not a piece was taken. Also, a lot of the code is dependent on "timers"

which just needs to go away. So I have a lot of refactoring to do. Hopefully this code will become beautiful by the time i finish with it. Also, a lot of the commands are based on

strings for now. In the future I may need to base them on enumerated values.

These are the three things I need to return from the player move function

1. If The player move took an opponent’s piece

2. The player move was valid and the board is updated

3. The player’s move was invalid.

Note that the chess engine already knows where the player moved from. Ideally it should pass in a "move" struct.

These are the things I need to return from the AI move function:

1. If the AI took a player's piece

2. If the AI resigned

3. Where the AI moved

-Currently there's an error when I attempt to put in an invalid move and call AI. I need to check that it's invalid.

-Things that may be useless and I should get rid of:

----Board History

----Timing!

Basically, I'm stripping down this guy's engine to the bone... it's good to strip down rather than build up, though.

\* Note: I need to figure out if I took a piece. The problem is the D function

\* actually moves every time it is searching. So I have to find some way of comparing the board

\* states before and after the move. Maybe this can be accomplished with the board pointer.

The alternative to returning the tookPiece flags is checking it manually and keeping my own copy of the board outside of the interface. That is a terrible idea as

there is already a copy of the board in memory.

# 8/12/2012

7:47 PM

I just spent two hours figuring out how to check for a taken piece. I realized the best way was finding the difference in number of pieces on the board between turns. It doesn't

require knowing much about the move that was taken and thus it can be generalized for both player and AI move. It is simply a linear search in the space of states which amounts to

little runtime.

Now I'm going to clean up the rest of the interface to be worthy of AVR. I'm going to get rid of all headers associated with windows and fix the maximum search depth to lower than 30.

I'll also remove anything about a "maximum game length". This should clean everything up very nicely, I just wanted to save it for last. Also, I need to decrease the size of the game board history

and this might require some reworking. I might need to make the game board history a circular buffer, which would be cool to implement. I sure am learning a lot.

8:27 PM

Wow, there was a LOT to clean up. I'm still using sprintf from stdio, hopefully I can use that in the AVR. Other than that things are looking about done.

# 8/16/2012

I attempted to port to AVR studio and found I had errors when trying to put my engine in a separate file from main. This is due to not understanding public/private headers. I was compiling inline

functions twice with using only public headers. Now I have created a private and public header to chess engine.

I found that my engine was able to compile and run on the simulator. It took up about 1KB of data memory (out of 8k) and about 6.1% of program memory (totalling about 8KB). This means that the program

is very low footprint which is good. We still need a big MCU for pin count but it looks like we could have done I2C.

8 PM:

Found that the simulator is running extremely slowly because it simulates the timers and therefore does not run in real time. Source: http://www2.tech.purdue.edu/ecet/courses/ecet309/Reference\_Materials/Simulation\_AVR\_Studio\_4.pdf

If i want to check the timing I need to try blinking some LEDs and create a test run.

8:10

I did a test where I blink all LEDs then wait 4 seconds and turn them off. From that point, I do one run of the simulator and then blink the LEDS every 2 seconds. The test ran perfectly and very quickly.

My second test is where I issue a series of 3 commands to the board. A2a4, a4a5, and a5a6. When I reach a6 I should get my piece taken. If an enemy AI move causes a piece to be taken, the lights should blink slowly.

If an enemy AI move does not, the lights should blink quickly.

I ran the test and on the third move the enemy properly captures the piece. The think time was about 2-3 seconds. That’s not bad. I should do a test later to improve the time.

# 8/21/2012

I’ve completed the chess engine interface and it works on the microcontroller. Now I have to begin the I/O system drivers. The most important systems are the motor controller and the human interface. I’ll be getting a motor control prototype from Paul on Thursday to begin messing with. Once I can move a couple motors and the claw, I need to perform calibration tests on the constructed platform. The platform will be constructed by Paul soon.

I have figured out that the board will be programmed with PDI since it’s incredibly easy to make a PDI programming interface on a PCB. I have seen a lot of header boards with PDI ports with the 128A1. I am getting an AVR Programmer from Olimex that is compatible with MKII.

The user interface consists of a demultiplexed button grid and a LCD on each side. The button grid should involve timing enables and polling from pins. Not much theory to worry about… all implementation specific. I’m not too worried. That doesn’t take away from the fact that we need to *order* the parts ASAP and begin prototyping them.

The LCD screens require a serial connection (or rather, I decided to use serial enabled LCDs). This means I need to attempt serial communication on my board. I have an old Arduino and I’m going to attempt to connect to it via the A1 over a serial connection and spit it out to the serial monitor.

Actually, before I bother with that, I found this guide: <http://www.atmel.com/Images/doc8319.pdf> Apparently I can communicate with myself in Loopback mode to test it. So I checked out the code and I found that I could loop back. And communicate using USART. First we specify that the pins 3 and 2 are Out/in, respectively. We user PORTD.DIRSET PIN#\_bm to set a output direction to 1, and DIRCLR to set it to 0. Then we do the CTRLC of USARTD0 to 8 bit/ parity disabled. Then we set the baud rate with USARTD0.BAUDCTRLA. Then we do the CTRLB bit masks to set receive/transmit. The problem is we’re looping on the same USART. I’m going to try an experiment where I send on one USART and receive on another.

Okay cool. So it turns out this serial communication will be useful. Basically, I have created an arduino sketch that allows me to hook up the ground of my A1 to the ground of the arduino and my GPIO pin D3 to pin 2 of the arduino. Then I can open the serial monitor and receive messages over the arduino. The sketch will be in my main project directory

# Next time:

Todo: implement basic user interface using serial output instead of LCD screens and onboard Xmega buttons. Implement control flow logic and create stub motor driver interface.