UNIVERSITY OF SUSSEX

BlueSense2 with Arduino

Setup Guide and Implementation

Nofal Shehzad

This document contains details and ‘theory’ on how to integrate third party hardware and firmware, and then looks at the implementation of Arduino compatibility for the BlueSense2. A second guide called ‘setup_basic.pdf’ is a short guide on how to install the files and program the device.

TABLE OF CONTENTS

Introduction

This document contains specifics on the setup files and packages used to integrate the BlueSense2 board with Arduino IDE. A second, simplified setup guide called *setup_basic* exists for a basic setup (see Appendix B), and allows users to download and place the files in the relevant place and follow a simple set of instructions in order to program the device with basic user code. This guide too would need to be expanded once the project progresses and more functionality is added via the IDE.

However, this specific guide goes in to details about the setup files used, how and why they were configured the way they are. And hopefully, this guide will also help inform on how to go about expanding them, making changes, updating versions, and a little bit on IDE versions is also discussed.

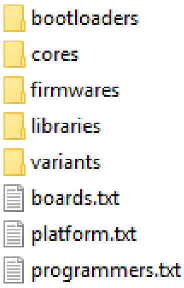
All setup files are available here:
https://github.com/NofalShehzad/BlueSense2-with-Arduino

Adding 3rd Party Hardware Specs to the Arduino IDE

Depending on the version of Arduino IDE being used, there exist some resources on how to go about integrating third party hardware. However, with this project, there is the added challenge of integrating existing firmware in c, as well as said third party hardware. And then there is a further challenge of using previously defined functions in the firmware in place of the empty standard Arduino cores definitions.

For basic 3rd party hardware integration. As of Arduino 1.5.x and newer (current version 1.8.1), there are number of different steps to integration. The version 1.8.1 uses avr-gcc version 4.9.2.

- The hardware folder which contains all board definitions and standard arduino functions needs to be modified.
- Within, the hardware folder (exact directory: `..\Arduino\hardware\arduino\avr`), the files `boards.txt, platform.txt` and `programmers.txt` would need to be edited to define the architecture.
- The file *boards.txt* contains the board specific definitions, board name and parameters. As standard there are a number of boards already defined in boards.txt as standard with Arduino. Third party board definitions would need to be appended to the file, or alternatively a simple `boards.txt` containing only the third party board definitions can be used to replace all other board definitions, however this will mean that other boards won't be accessible through the IDE anymore.
- The `platform.txt` file contains the definitions for the CPU architecture, build process and compiler details. This will need to edited with the specifications outlines in the current version firmware's `makefile`. Note, that doing this however will change the build process for ALL other boards that use the IDE.
- The `programmers.txt` file contains definitions for external programmers, and bootloaders. However, for now with no bootloader this will not need to be edited. As by standard it contains the definitions for AVRISP mkII and Atmel Studio is used to upload the resultant hex file.



bootloaders
cores
firmwares
libraries
variants
boards.txt
platform.txt
programmers.txt

Fig 1. The contents of the hardware folder

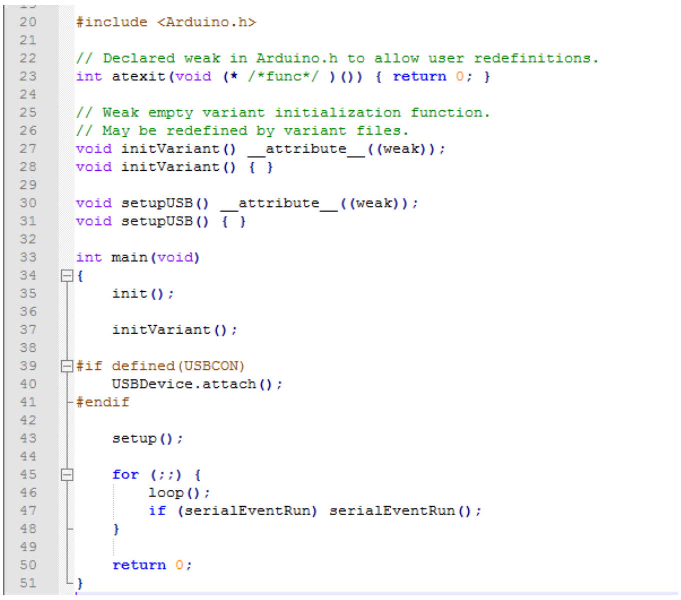
In addition to the architectural specifications in that single directory. The subdirectories cores and variants need changing in order to make use of Arduino functions and define pins according the schematic of the bluesense2.

The cores folder contains Arduino functions like `digitalRead()/digitalWrite(), analogRead()/analogWrite(), delay()`, as well as Serial Comms functions such as `Serial.print()` and `Serial.println().` These definitions are mostly in c++, and the folder also contains a wiring file, and a main file.

Main File

The main file, `main.cpp` in the folder `"\Arduino\hardware\arduino\avr\cores\arduino\main.cpp"` is the file that takes the user code from `void setup()` and `void loop()` in the user sketch and uses an initial `init()` method, however this method will be overwritten by the firmware's own `init()`.

The `setup()` function on line 43 corresponds to the setup function in the user Sketch. And `loop()` on line 46 corresponds to the loop function in the user sketch, the for loop is what makes the user code repeat.



```
20  #include <Arduino.h>
21
22  // Declared weak in Arduino.h to allow user redefinitions.
23  int atexit(void (* /*func*/ )()) { return 0; }
24
25  // Weak empty variant initialization function.
26  // May be redefined by variant files.
27  void initVariant() __attribute__((weak));
28  void initVariant() { }
29
30  void setupUSB() __attribute__((weak));
31  void setupUSB() { }
32
33  int main(void)
34  {
35      init();
36
37      initVariant();
38
39  #if defined(USBCON)
40      USBDevice.attach();
41  #endif
42
43      setup();
44
45      for (;;) {
46          loop();
47          if (serialEventRun) serialEventRun();
48      }
49
50      return 0;
51  }
```

Fig 2. Main.cpp in the cores folder, containing the loop() and setup() that user sketches reference.

Pin Configuration File

For all boards programmable with Arduino, a pin mapping file called `pins_arduino.h` needs to be created, inside the variants folder. For instance, the BlueSense2 board would need to placed inside a folder in the following directory:
`..\Arduino\hardware\arduino\avr\variants`

A new folder for the BlueSense2 needs to be created that corresponds to the path defined in `boards.txt` and in it needs to be placed the `pins_arduino.h` file of the board, such that the directory would show:
`..\Arduino\hardware\arduino\avr\variants\bluesensev2\pins_arduino.h`

And this `pins_arduino.h` file would contain definitions written using the BlueSense2 schematic.

Firmware Included as an Arduino 'Library'

The firmware as it stands comprises mainly of .c source files and .h header files. However, compilation issues, g++ compatibility issues meant that a minimized version had to be used. However, the all such files would be placed in the following directory inside a folder you can name 'BlueSense2' for example:

`..\Arduino\libraries\BlueSense2`

However, placing the firmware on its own in the directory and using it as is will cause many errors related to the fact that you'd be compiling c files with Arduino that uses a combination of c and c++.

Implementation

The guide above roughly outlines what needs to be added, changed and modified. The following text contains what work has been done, how it has been done and what there's left to do.

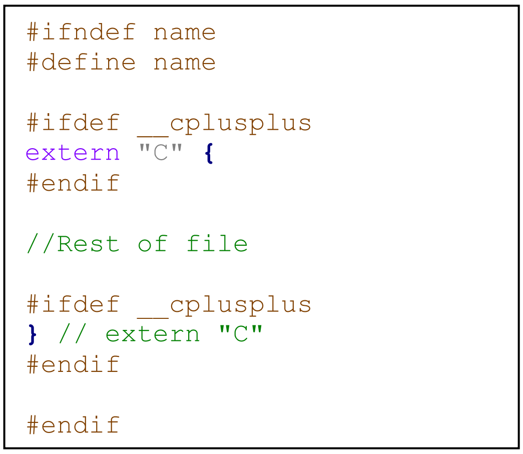
Integration of Firmware

The firmware when integrated as a whole from the BlueSense2 firmware repository, had some compilation and g++ compatibility issues, and therefore a simpler minimal firmware was used in its place comprising of the most basic functions of the device and an almost empty main file.

It was hoped that if this minimal version could be used effectively, it could be built upon to use more complex versions of the firmware. And there would be a process of integrating more of the hardware definitions along with this too.

However, using the .c source and .h header files in the minimized firmware, they need to be placed in a dedicated library folder that the IDE will use to import the firmware for every user sketch, allowing the user sketch to make use of the firmware's functions.

In order to avoid errors, due to compatibility issues, extern C blocks need to be added to all header files within the library folder containing the firmware.



```
#ifndef name
#define name

#ifdef __cplusplus
extern "C" {
#endif

//Rest of file

#ifdef __cplusplus
} // extern "C"
#endif

#endif
```

Fig 3. The code used to modify each header file from the firmware in the BlueSense2 library.

Other errors might still persist, such as two functions with separate uses and definitions causing a previously defined error. This can be fixed by renaming carefully the function taking into account where it's used.

KEYWORDS.TXT IN LIBRARIES

All Arduino libraries also have an optional *keywords.txt* in which methods, functions, literals and datatypes are stated in order to allow the IDE to highlight them orange so that the user can see a keyword in the editor.

The typical layout of the file is as follows:

```
########################################
# Datatypes (KEYWORD1)
########################################

datatype1   KEYWORD1
datatype2   KEYWORD1

//enter any datatype and then append KEYWORD1

########################################
# Methods and Functions (KEYWORD2)
########################################

function1   KEYWORD2
function2   KEYWORD2

//enter any function and then append with KEYWORD2

########################################
# Constants (LITERAL1)
########################################

costant1 LITERAL1
```

Fig 4. Shows an example keywords.txt file, the actual keywords.txt for the BlueSense2 library is too large to be displayed and doesn’t as easily convey how it is meant to be set up.

There file keywords.txt for the bluesense2 library was far too large to be included in this document. However, if the keywords.txt needs to include many functions or constants. It may be useful to use copy and paste the names, and use find and replace in a text editor to remove for instance the return types of functions.

And for functions placed in the document using copy and paste, in order to get rid of the brackets a regular expression with find and replace can be used:

Regex: “\((.*?)\)” to get rid of brackets or “\((.*?)\;” to get rid of function parenthesis and semi-colon “();”.

Hardware Integration

Hardware integration requires the contents of the hardware folder to be changed, specifically, *boards.txt, platform.txt, pins_arduino.h* to be added, and files in the cores folder to be modified.

Modifying boards.txt

The file *boards.txt* as discussed before contains the definitions for each board and allows the IDE to recognize the device , select and program it or compile for it.

The definitions for the BlueSense2 have been appended at the end of the file.

```
##############################################################

1284p.name=bluesenseV2
1284p.upload.protocol=stk500v1
1284p.upload.maximum_size=128000
1284p.upload.speed=115200
1284p.bootloader.low_fuses=0xFF
1284p.bootloader.high_fuses=0xD1
1284p.bootloader.extended_fuses=0xFF
1284p.bootloader.unlock_bits=0x3F
1284p.bootloader.lock_bits=0x0F
1284p.build.mcu=atmega1284p
1284p.build.f_cpu=11059200L
1284p.build.core=extra-cores
1284p.build.variant=bluesensev2
```

Fig 5. Shows part of the contents of boards.txt, containing the BlueSense2 definitions.

The label '1284p' is not necessary and can be changed to anything, but must be changed for all lines. The first line defined the name as it would appear in the boards manager of the IDE. So, under Tools=>Boards, the bluesense2 will appear under contributed boards as 'bluesenseV2'. No bootloader is actually used through the IDE but some of the definitions are there, an actual bootloader would require another line '*1284p.bootloader.path=…*' to be added, which specifies the path to the bootloader.

The '*1284p.build.core=extra-cores'* defines the path and folder for cores used. Here extra-cores is used because modification have been made from standard arduino cores. And similarly, the '*1284p.build.variant=bluesensev2'* specifies the path to the *pins_arduino.h* file specific to the bluesense2.
Thus, the last two lines specify the path to the cores and *pins_arduino.h* specific to Arduino from where the boards.txt file is located. Any changes would need to be added to the file accordingly.

Cores, contains the main file and the standard Arduino definitions and functions. In order to integrate the device and the firmware correctly, the 'extra-cores' for the BlueSense2 device needs to be expanded upon. As it stands, the only changes made are to the '*Print.h'* file in which definitions are added to allow the use of '*Serial.printf()'* as opposed to just *print* and *println*.

Build Process Editing

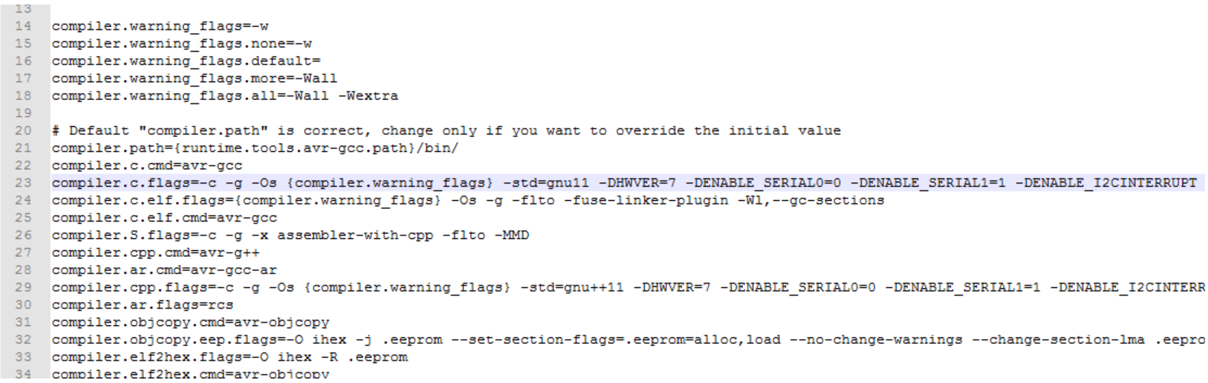
Editing the build process is required in order for the firmware to compile properly, since the makefile for the firmware isn't used directly.

The parameters within need to be specified where the IDE uses them.

The file *platform.txt* contains all the information for the build and compilation process and had the following parameters appended to one particular line.

```
-DHWVER=7
-DENABLE_SERIAL0=0
-DENABLE_SERIAL1=1
-DENABLE_I2CINTERRUPT
-DFIXEDPOINTQUATERNION=0
-DFIXEDPOINTQUATERNIONSHIFT=0
-DENABLEQUATERNION=1
-DENABLEGFXDEMO=1
-DENABLEMODECOULOMB=1
-DBOOTLOADER=0
-D__DELAY_BACKWARD_COMPATIBLE__
```

Fig 6-7. Shows the parameters from the makefile and how they're added to platform.txt



```

compiler.warning_flags=-w
compiler.warning_flags.none=-w
compiler.warning_flags.default=
compiler.warning_flags.more=-Wall
compiler.warning_flags.all=-Wall -Wextra

# Default "compiler.path" is correct, change only if you want to override the initial value
compiler.path={runtime.tools.avr-gcc.path}/bin/
compiler.c.cmd=avr-gcc
compiler.c.flags=-c -g -Os {compiler.warning_flags} -std=gnu11 -DHWVER=7 -DENABLE_SERIAL0=0 -DENABLE_SERIAL1=1 -DENABLE_I2CINTERRUPT
compiler.c.elf.flags={compiler.warning_flags} -Os -g -flto -fuse-linker-plugin -Wl,--gc-sections
compiler.c.elf.cmd=avr-gcc
compiler.S.flags=-c -g -x assembler-with-cpp -flto -MMD
compiler.cpp.cmd=avr-g++
compiler.ar.cmd=avr-gcc-ar
compiler.cpp.flags=-c -g -Os {compiler.warning_flags} -std=gnu++11 -DHWVER=7 -DENABLE_SERIAL0=0 -DENABLE_SERIAL1=1 -DENABLE_I2CINTERR
compiler.ar.flags=rcs
compiler.objcopy.cmd=avr-objcopy
compiler.objcopy.eep.flags=-O ihex -j .eeprom --set-section-flags=.eeprom=alloc,load --no-change-warnings --change-section-lma .eepro
compiler.elf2hex.flags=-O ihex -R .eeprom
compiler.elf2hex.cmd=avr-objcopy
```

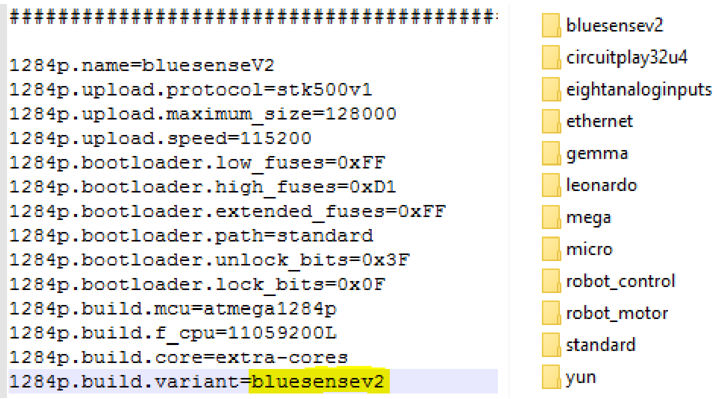
This allows the firmware to be compiled with the parameters defined in the makefile. The verbose compilation process now will display these parameters while the IDE compiles. For older versions of the device and firmware, *DHWVER* would need to be changed.

The file platform.txt also contains information on the upload process, and tools, however the IDE isn't yet used for upload, only compilation (verify).

Pins_arduino.h

This file contains the pin mapping for each device, in terms of directory structure, the *pins_arduino.h* files are located in separate folder within *"..\Arduino\hardware\arduino\avr\variants".*

Each folder contains a singular *pins_arduino.h* file for each compatible board, the name of the folder should be the same as the path specified in boards.txt as '1284.build.varaints=bluesensev2':



```
##########################################

1284p.name=bluesenseV2
1284p.upload.protocol=stk500v1
1284p.upload.maximum_size=128000
1284p.upload.speed=115200
1284p.bootloader.low_fuses=0xFF
1284p.bootloader.high_fuses=0xD1
1284p.bootloader.extended_fuses=0xFF
1284p.bootloader.path=standard
1284p.bootloader.unlock_bits=0x3F
1284p.bootloader.lock_bits=0x0F
1284p.build.mcu=atmega1284p
1284p.build.f_cpu=11059200L
1284p.build.core=extra-cores
1284p.build.variant=bluesensev2
```

```
bluesensev2
circuitplay32u4
eightanaloginputs
ethernet
gemma
leonardo
mega
micro
robot_control
robot_motor
standard
yun
```

Fig 8. Illustrates how the boards.txt definitions for cores and variant describe the path to the folders containing pins_arduino and cores files.

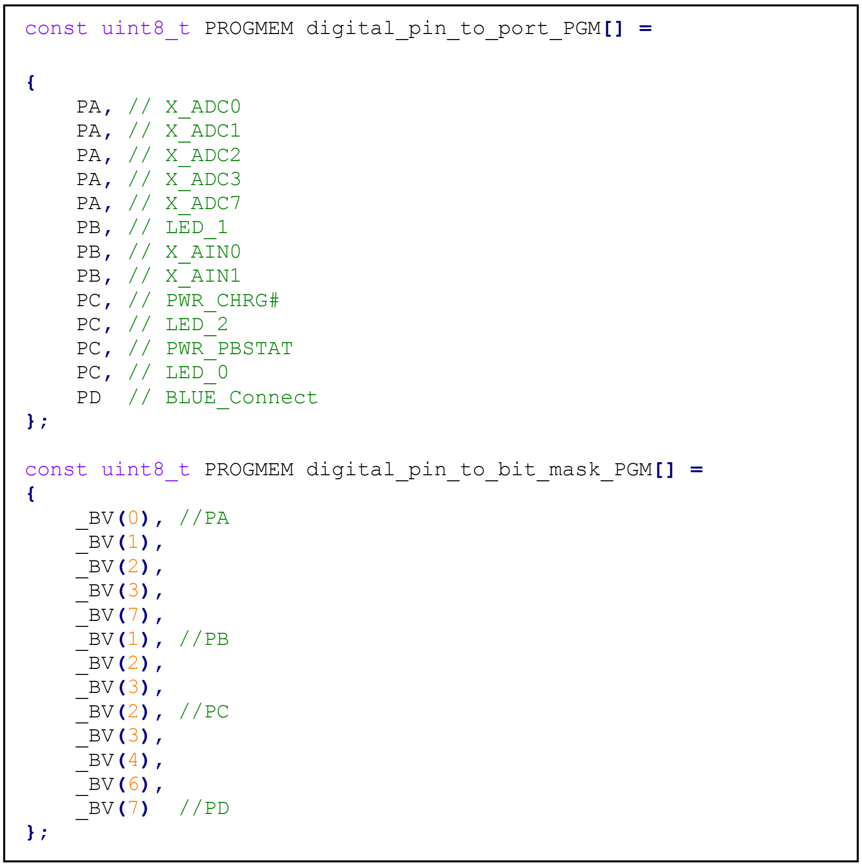
The contents of *pins_arduino.h* for the bluesense version was made using a modified version of an implementation of the 1284p, all definitions within are the same except for the following arrays which correspond to one another:

- *PROGMEM digital_pin_to_port_PGM[]*
- *PROGMEM digital_pin_to_bit_mask_PGM[]*
- *PROGMEM digital_pin_to_timer_PGM[]*

Of the standard file, these three arrays were emptied, and then written over using some of the pins of the device that are useful for testing the minimal firmware. The following pins are currently defined:

- *X_ADC0-3*
- *X_ADC-7*
- *LED_0-2*
- *X_AIN0-1*
- *PWR_CHRG#*
- *PWR_PBSTAT*
- *BLUE_Connect*

These were defined using the schematic (see Appendix A), the position of pins in the schematic PA-PD were used to order the pins defined in *pins_arduino.h*.



```
const uint8_t PROGMEM digital_pin_to_port_PGM[] =

{
    PA, // X_ADC0
    PA, // X_ADC1
    PA, // X_ADC2
    PA, // X_ADC3
    PA, // X_ADC7
    PB, // LED_1
    PB, // X_AIN0
    PB, // X_AIN1
    PC, // PWR_CHRG#
    PC, // LED_2
    PC, // PWR_PBSTAT
    PC, // LED_0
    PD  // BLUE_Connect
};

const uint8_t PROGMEM digital_pin_to_bit_mask_PGM[] =
{
    _BV(0), //PA
    _BV(1),
    _BV(2),
    _BV(3),
    _BV(7),
    _BV(1), //PB
    _BV(2),
    _BV(3),
    _BV(2), //PC
    _BV(3),
    _BV(4),
    _BV(6),
    _BV(7)  //PD
};
```

Fig 9. Contents of pins_arduino.h, two of the three modified arrays containing the pins corresponding to the schematic. (see Appendix A)

The order of the pins defined in first array listed correspond to the pinout numbers the Arduino uses. For instance, the 5th element of that array is *'PB, //LED_1'*.

So in order to use that for an Arduino sketch, the following code can be used with the number 5, corresponding to the LED pin as defined as the 5th element listed:

```
int led1=5;


void setup() {
   pinMode(led1, OUTPUT);
}

void loop() {
    digitalWrite(led1, HIGH);
    _delay_ms(1000);
    digitalWrite(led1, LOW);
    _delay_ms(1000);
}
```

Fig 10. An example user sketch that toggles an LED with the pin number defined in pins_arduino.h

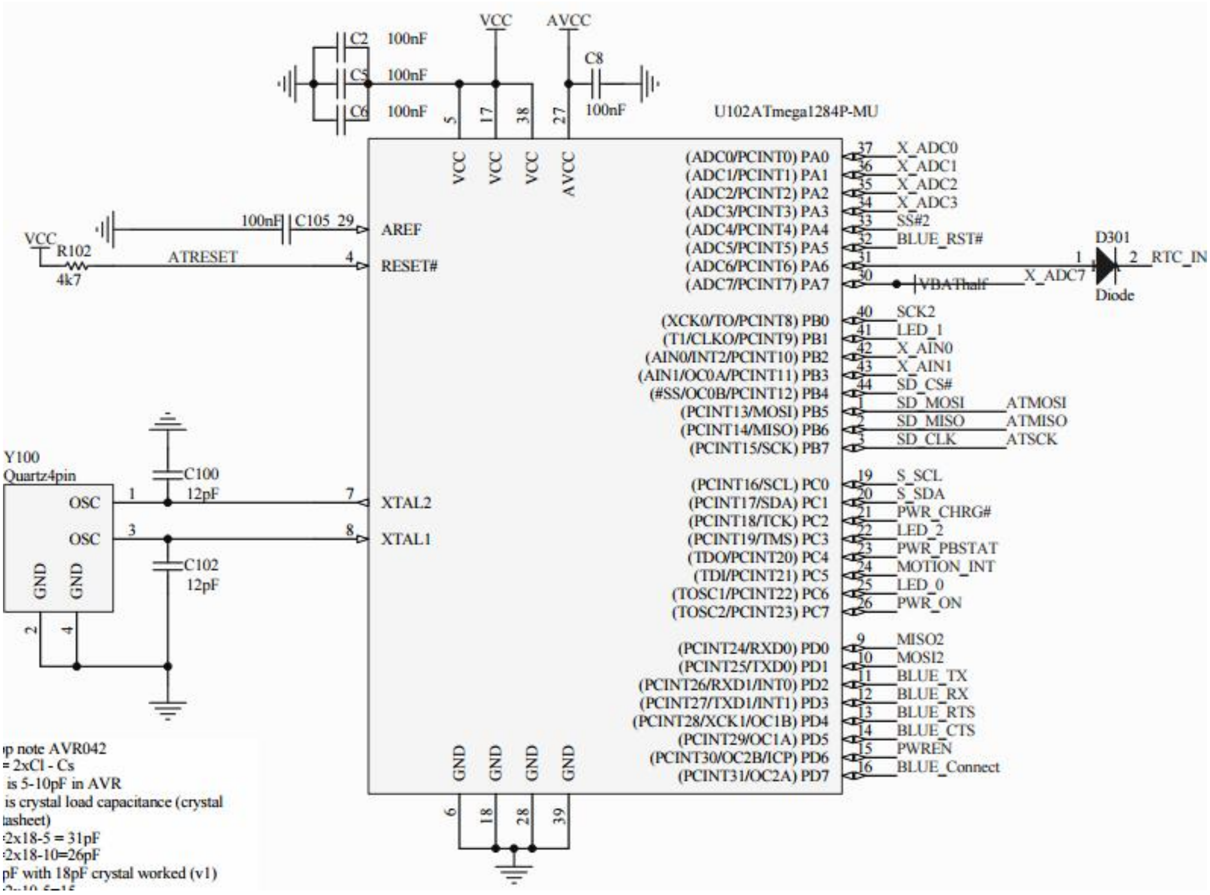
This code would toggle the *LED_1* on and off every second. As more pins are added to the file, the order in which these are defined and are therefore accessed changes.

The second array where '*BV*' stands for Bit Value, lists the values by Ports, starting from 0 to the number of pins of Port A, and then does the same for Ports B, C and D. And if more pins are defined, the second array would be needed to be expanded upon accordingly. The third array only has *'NOT_A_TIMER'* for every pin in the other two as it stands so far.

Appendices

Appendix A

An updated schematic of the BlueSense2 device used to define pins in pins_arduino.h



VCC
AVCC
C2 100nF
C5 100nF
C6 100nF
C8 100nF
5
17
38
27
U102ATmega1284P-MU
VCC
VCC
VCC
AVCC
100nF C105 29
AREF
VCC
R102
4k7
ATRESET
4
RESET#
(ADC0/PCINT0) PA0 37 X_ADC0
(ADC1/PCINT1) PA1 36 X_ADC1
(ADC2/PCINT2) PA2 35 X_ADC2
(ADC3/PCINT3) PA3 34 X_ADC3
(ADC4/PCINT4) PA4 33 SS#2
(ADC5/PCINT5) PA5 32 BLUE_RST#
(ADC6/PCINT6) PA6 31
(ADC7/PCINT7) PA7 30 ~~VBAThalf~~ X_ADC7
D301
1
2
RTC_IN
Diode
(XCK0/TO/PCINT8) PB0 40 SCK2
(T1/CLKO/PCINT9) PB1 41 LED_1
(AIN0/INT2/PCINT10) PB2 42 X_AIN0
(AIN1/OC0A/PCINT11) PB3 43 X_AIN1
(#SS/OC0B/PCINT12) PB4 44 SD_CS#
(PCINT13/MOSI) PB5 1 SD_MOSI ATMOSI
(PCINT14/MISO) PB6 2 SD_MISO ATMISO
(PCINT15/SCK) PB7 3 SD_CLK ATSCK
Y100
Quartz4pin
C100
12pF
OSC 1
OSC 3
GND
GND
2
4
7 XTAL2
8 XTAL1
C102
12pF
(PCINT16/SCL) PC0 19 S_SCL
(PCINT17/SDA) PC1 20 S_SDA
(PCINT18/TCK) PC2 21 PWR_CHRG#
(PCINT19/TMS) PC3 22 LED_2
(TDO/PCINT20) PC4 23 PWR_PBSTAT
(TDI/PCINT21) PC5 24 MOTION_INT
(TOSC1/PCINT22) PC6 25 LED_0
(TOSC2/PCINT23) PC7 26 PWR_ON
(PCINT24/RXD0) PD0 9 MISO2
(PCINT25/TXD0) PD1 10 MOSI2
(PCINT26/RXD1/INT0) PD2 11 BLUE_TX
(PCINT27/TXD1/INT1) PD3 12 BLUE_RX
(PCINT28/XCK1/OC1B) PD4 13 BLUE_RTS
(PCINT29/OC1A) PD5 14 BLUE_CTS
(PCINT30/OC2B/ICP) PD6 15 PWREN
(PCINT31/OC2A) PD7 16 BLUE_Connect
GND
GND
GND
GND
6
18
28
39
p note AVR042
= 2xCl - Cs
is 5-10pF in AVR
is crystal load capacitance (crystal
tasheet)
2x18-5 = 31pF
2x18-10=26pF
pF with 18pF crystal worked (v1)

Appendix B

The link to the setup files and the setup_basic.pdf guide for the BlueSense2 with Arduino from the github repository:

https://github.com/NofalShehzad/BlueSense2-with-Arduino