Embedded System Design Documentation

1. Target Embedded System

**Power**

The target microcontroller ATmega328p requires power input between 3V – 6.6V. We used MCP18XX regulator which gives fixed output voltage of 3.3V. This choice was done to be compatible with MSP432 SPI Port, as MSP432 operates on 3.3V.

Talk about power consumption by the devices I would be using

**Why ATmega328p?**

ATmega328p is a microcontroller built on AVR RISC architecture. This platform is new to us, and this class of microcontroller is widely used in Arduino boards, for which heavy documentation is available in case there is any problem.

Links to reference I read

**What did we learn about AVR?**

ATmega328p has an AVR Core with registers. It has 32kB of Flash memory and 6kB of SRAM to use during program run. It also hosts 2kB of EEPROM for code or data storage. Flash Memory can be coded through Serial and Parallel Programming. However, we decided to go with Serial Programming as it requires less hardware support.

AVR Core, AVR Memory, AVR Clock, AVR Peripherals – things I read

**ISP Protocol**

ISP Protocol for ATmega328p is simple. It is built on SPI. SPI module in ATmega328p activates as reset is active. Once reset is active, wait for 20 ms, and then send programming enable command. If the device is in sync, then it will echo back “53” which is second byte of Programming Enable Command. Subsequent commands follow suit, and the reset is held active till the programming option is finished.

* List the commands in appendix
* How did we implement in the program

1. Programmer

**MSP432P401R**

We are using MSP432P401R as a programmer. Leveraging the SPI peripheral, and understanding how it operates, we are controlling the ATmega328p pins for Reset, SCK, MOSI, MISO to emulate the programming sequence and algorithm required for ATmega328p.

**SPI Peripheral**

SPI Peripheral in MSP432P401R is convenient, and enough documentation is provided to get it started. We went through the peripheral registers, and designed a library to initialize SPI Peripheral, send and receive data through polling.

**ISP Command Interface**

SPI library is used by ISP Command Interface which manipulates Reset, SCK, MOSI, and reads through MISO. ISP Command Interface is used by Programmer State Machine to serially download code into ATmega328p. This element has been designed to perform ISP operations on ATmega328p.

**Hex Parsing**

Hex file is transmitted by Transmitter Device through Bluetooth. This hex file shall be parsed to deduce required ISP Commands that we will be needing. This needs understanding of Intel Hex Format – which we are going to use, and memory map of ATmega328p Microcontroller.

**Bluetooth Data Reception**

We are using HC-05 Module to communicate through Bluetooth to the Transmitter. This module is used in AT mode to set the role as a slave device, and interfaces with MSP432 using UART. This module deals with data reception, and transfers the data over to MSP432.

**CRC/Checksum**

As, the data is program data, and very crucial. We have Checksum in place to detect any error in transmission. As the data travels through multiple communication channels, we thought it would be better to ensure that the data is valid by using CRC32 peripheral in MSP432.

**State Machine**

**\_\_Diagram\_\_HERE\_\_**

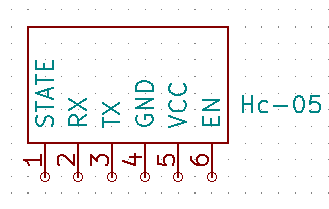
1. Transmitter

**Bluetooth Side**

One of our goals in this project was to experiment with a wireless protocol. There were many options for us to choose from but we mainly debated between using WiFi and Bluetooth. WiFi would have been a good option had we wanted to provide internet capability to our project. Since we just wanted to replace UART wires, Bluetooth was cheap solution.

We chose the ubiquitous HC-05 module from DSD tech in our project. The HC-05 is one of the most popular modules that provide Serial Port Profile, which is essentially UART without the wires. This module provided us with Bluetooth version V2.0 + EDR which theoretically provides data rate of 3 Mbps, which was more than enough considering we planned on transferring data of a few Kilobytes.

The main issue we faced in using this module was inadequate documentation. We needed to configure our modules using AT Commands. The module needs to be in configuration mode, in order to change the baud rate, the name of the module, the password and to pair and bind the device to other Bluetooth modules. However, it was not clear how to switch to configuration mode. The documentation did not clearly explain how to do this.



After scouring the internet, we understood the procedure which was to provide power after pulling up the EN pin. Conversely, to enter the communication the EN has to be pulled low before providing power to the module. The HC-05 communicates across the UART at 38400 baud rate in configuration mode and 9600 baud rate in communication mode, another key piece of information which was not conveyed well in the documentation and therefore caused some problem when we tried to configure the modules.

After solving this, we configured the role of the two modules using AT commands as slave and master and made the following configurations.

1. Changed the baud rate to 38400
2. Changed the connection mode to connect the module to one address
3. Acquired the address of each module
4. Bound the two modules to each other

These changes were important because we did not want any other device to accidently or maliciously gain access to our programmer.

Once we got this out of the way, the module was very easy to use. We used two UART ports on the MSP432. The EUSCI\_A0 port was used to communicate with our PC and EUSCI\_A2 to communicate with the Bluetooth module, so we could easily test whether data was being exchanged correctly.

**Anticipated Error Cases during communication**

During the design phase of our project, we listed out problems that we could face in using a wireless protocol which are:

1. Data corruption during transmission
2. Incomplete transmission of data
3. Malicious/Accidental Bluetooth access to our programmer

To solve the problem of data corruption during transmission we decided to use some kind of a checksum. We first used an algorithm which found the two’s complement of the sum of our data bits and appended it to our data, the receiver then finds the two’s complement of the data including the checksum and if that’s zero then transmission was correct. However, we understood later that, simple algorithms such as this may not detect corruption that may happen to several bits at once. We decided to use CRC as it is a common checksum algorithm used and moreover MSP432 provides a 16 bit CRC module. The CRC16 is based on the polynomial given in the CRC‑CCITT standard with f(x) = x16 + x12 + x5 + 1.

The module is used in the following manner,

1. A SEED value is written to the CRC Initialisation and Result register (CRC16INRES)
2. The data is fed to the CRC Data Input (CRC16DI)
3. The CRC result is acquired from the CRC Initialisation and Result register (CRC16INRES)

The CRC result is appended to the data that we send from the master device to the slave device where the programmer performs CRC as described above on the data received and compares it to the CRC result received from the master. If they are equal it carries on with programming the target but, if they are not it sends back a message indicating that a CRC error has taken place. The master then resends the data.

Another, problem that we though could happen was incomplete transmission of data. As our project is used for programming a microcontroller, the programmer must receive the entire, uncorrupted hex file. The scenario we had in mind was – what if the power is cut off on the master side during mid-transmission and the entire hex file is not received?

To solve this problem we designated (-1) as our end of file indicator (EOF). The programmer does not consider the transmission to be complete until it receives the (EOF). This (EOF) also helps in keeping a count of the number of bytes received.

As described earlier the devices were bound to each other using AT commands in the configuration mode, to prevent someone from accidently or maliciously gaining access over our programmer.