University of California, Riverside

Lab #6 Report

Serial Communication (UART, SPI, and I2C)

EE 128 Section 21, Fall 2025

Name: Muneer Al Jufout

Student ID: 862410258

Lab Partner: Sameer Anjum

Lab Partner ID: 862422332

TA: Johnson Zhang

Abstract

The lab explained UART, I2C, and SPI on the K64F board. First, we used I2C to read data from the accelerometer and magnetometer and send that data to the computer via UART; then, we established SPI between the K64F and an Arduino Uno to send data and view it in the Arduino serial monitor. We measured the time delay from sending and receiving a message by raising GPIO pins and checking the timing on the oscilloscope.

Experiment System Specification

For this lab, we used Port B to generate the timing pulse and the SPI pins on Ports C and D to send data from the K64F to the Arduino. The Arduino received the data and produced the end pulse so we could measure the latency on the oscilloscope.

Flowchart

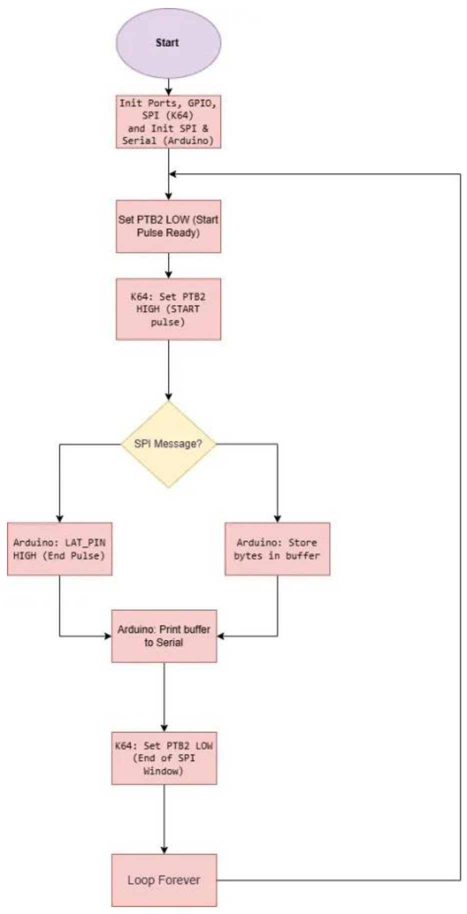
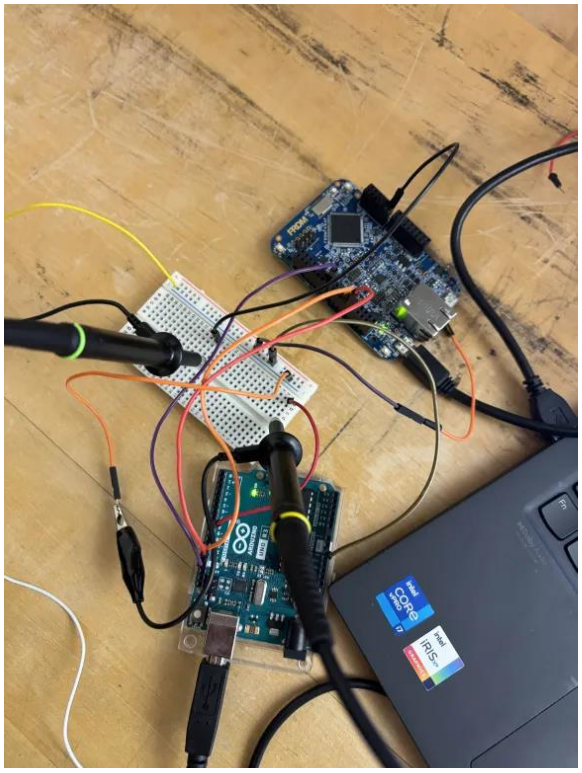

Start
Init Ports, GPIO, SPI (K64) and Init SPI & Serial (Arduino)
Set PTB2 LOW (Start Pulse Ready)
K64: Set PTB2 HIGH (START pulse)
SPI Message?
Arduino: LAT_PIN HIGH (End Pulse)
Arduino: Store bytes in buffer
Arduino: Print buffer to Serial
K64: Set PTB2 LOW (End of SPI Window)
Loop Forever

Fig. 1: The Program Flowchart

Hardware Design


a)

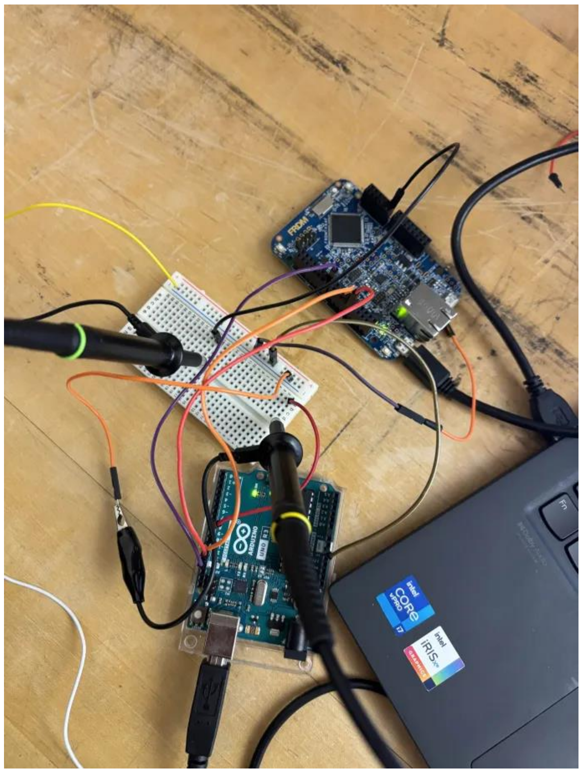

b)

Fig. 2: Photos of the Setup

Schematic Diagram

This schematic included:

- MK64FN1M0VLL12 microcontroller (FRDM-K64F)
- Ground (GND)
- ATmega328P (Arduino)

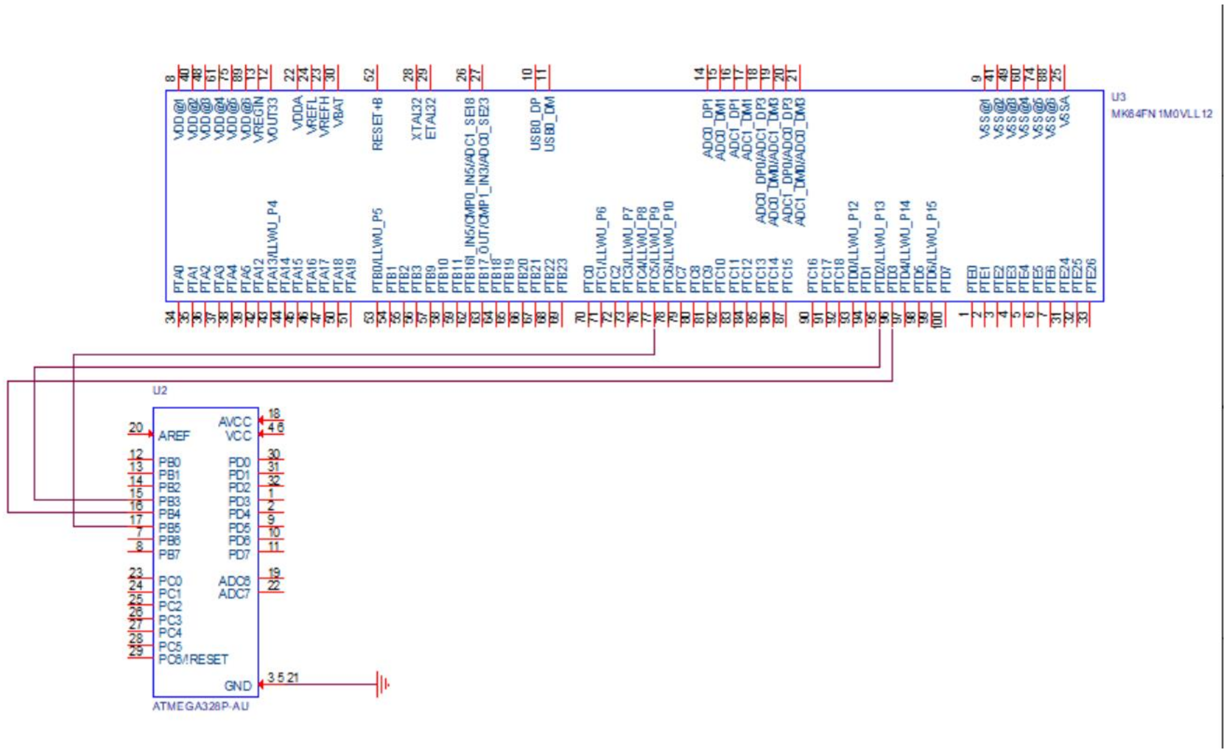


Fig. 3: Schematic Diagram

High-Level Description

The program reads data from the onboard accelerometer and magnetometer over I2C and then transfers that data over SPI to the Arduino. The K64F raises a GPIO pin before each SPI transfer, and the Arduino raises its pin on receipt of the end of a message. These two pulses are then used on the oscilloscope to measure latency. The Arduino prints such received data to the serial monitor, while the K64F continues in the read-and-send cycle.

Program Listing (Main Sections)

FRDM-K64F Code

Part 2 (K64):

```
/* ###################################################################
**     Filename    : main.c
**     Project     : Lab6_Part2
**     Processor   : MK64FN1M0VLL12
**     Version     : Driver 01.01
**     Compiler    : GNU C Compiler
**     Date/Time   : 2019-11-03, 17:50, # CodeGen: 0
**     Abstract    :
**         Main module.
**         This module contains user's application code.
**     Settings    :
**     Contents    :
**         No public methods
**
** ###################################################################*/
/*!
 ** @file main.c
 ** @version 01.01
 ** @brief
 **         Main module.
 **         This module contains user's application code.
 */
/*!
 **  @addtogroup main_module main module documentation
 **  @{
 */
/* MODULE main */


/* Including needed modules to compile this module/procedure */
#include "Cpu.h"
#include "Events.h"
#include "Pins1.h"
#include "FX1.h"
#include "GI2C1.h"
#include "WAIT1.h"
#include "CI2C1.h"
#include "CsIO1.h"
#include "IO1.h"
#include "MCUC1.h"
#include "SM1.h"
/* Including shared modules, which are used for whole project */
#include "PE_Types.h"
#include "PE_Error.h"
#include "PE_Const.h"
#include "IO_Map.h"
#include "PDD_Includes.h"
#include "Init_Config.h"
/* User includes (#include below this line is not maintained by Processor Expert) */

/*lint -save  -e970 Disable MISRA rule (6.3) checking. */

unsigned char write[512];
int main(void)
/*lint -restore Enable MISRA rule (6.3) checking. */
{
  /* Write your local variable definition here */
```

```
/*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
PE_low_level_init();
/*** End of Processor Expert internal initialization.                    ***/
/* Write your code here */
uint32_t delay;
uint8_t ret, who;
int8_t temp;
int16_t accX, accY, accZ;
int16_t magX, magY, magZ;

int len;
LDD_TDeviceData *SM1_DeviceData;
SM1_DeviceData = SM1_Init(NULL);

printf("Hello\n");

FX1_Init();

for(;;) {
   // get WHO AM I values
   if (FX1_WhoAmI(&who)!=ERR_OK) {
      return ERR_FAILED;
   }
   printf("Who Am I value in decimal \t: %4d\n",who);
   len = sprintf(write, "Who Am I  value in decimal \t: %4d\n",who);
   SM1_SendBlock(SM1_DeviceData, &write, len);
   for(delay = 0; delay < 300000; delay++); //delay

   // get raw temperature values
   if (FX1_GetTemperature(&temp)!=ERR_OK) {
      return ERR_FAILED;
   }
   printf("RAW Temperature value in decimal \t: %4d\n",temp);
   len = sprintf(write, "RAW Temperature value in decimal \t: %4d\n",temp);
   SM1_SendBlock(SM1_DeviceData, &write, len);
   for(delay = 0; delay < 300000; delay++); //delay

   // Set up registers for accelerometer and magnetometer values
   if (FX1_WriteReg8(FX1_CTRL_REG_1, 0x00) != ERR_OK) {
      return ERR_FAILED;
   }
   if (FX1_WriteReg8(FX1_M_CTRL_REG_1, 0x1F) != ERR_OK) {
      return ERR_FAILED;
   }
   if (FX1_WriteReg8(FX1_M_CTRL_REG_2, 0x20) != ERR_OK) {
      return ERR_FAILED;
   }
   if (FX1_WriteReg8(FX1_XYZ_DATA_CFG, 0x00) != ERR_OK) {
      return ERR_FAILED;
   }
   if (FX1_WriteReg8(FX1_CTRL_REG_1, 0x0D) != ERR_OK) {
```

```
            return ERR_FAILED;
        }

        // Get the X Y Z accelerometer values
        accX = FX1_GetX();
        accY = FX1_GetY();
        accZ = FX1_GetZ();
        printf("Accelerometer value \tX: %4d\t Y: %4d\t Z: %4d\n", accX, accY, accZ);
        len = sprintf(write, "Accelerometer value \tX: %4d\t Y: %4d\t Z: %4d\n", accX, accY, accZ);
        SM1_SendBlock(SM1_DeviceData, &write, len);
        for(delay = 0; delay < 300000; delay++); //delay

        // Get the X Y Z magnetometer values
        if (FX1_GetMagX(&magX)!=ERR_OK) {
            return ERR_OK;
        }
        if (FX1_GetMagY(&magY)!=ERR_OK) {
            return ERR_OK;
        }
        if (FX1_GetMagZ(&magZ)!=ERR_OK) {
            return ERR_OK;
        }
        printf("Magnetometer value \tX: %4d\t Y: %4d\t Z: %4d\n", magX, magY, magZ);
        len = sprintf(write, "Magnetometer value \tX: %4d\t Y: %4d\t Z: %4d\n", magX, magY, magZ);
        SM1_SendBlock(SM1_DeviceData, &write, len);
        for(delay = 0; delay < 300000; delay++); //delay
    }


    /* For example: for(;;) { } */

    /*** Don't write any code pass this line, or it will be deleted during code generation. ***/
  /*** RTOS startup code. Macro PEX_RTOS_START is defined by the RTOS component. DON'T MODIFY THIS
CODE!!! ***/
  #ifdef PEX_RTOS_START
    PEX_RTOS_START();                  /* Startup of the selected RTOS. Macro is defined by the RTOS component. */
  #endif
  /*** End of RTOS startup code.  ***/
  /*** Processor Expert end of main routine. DON'T MODIFY THIS CODE!!! ***/
  for(;;){}
  /*** Processor Expert end of main routine. DON'T WRITE CODE BELOW!!! ***/
} /*** End of main routine. DO NOT MODIFY THIS TEXT!!! ***/

/* END main */
/*!
 ** @}
 */
/*
 ** ###################################################################
 **
 **     This file was created by Processor Expert 10.4 [05.11]
 **     for the Freescale Kinetis series of microcontrollers.
 **
 ** ###################################################################
 */
```

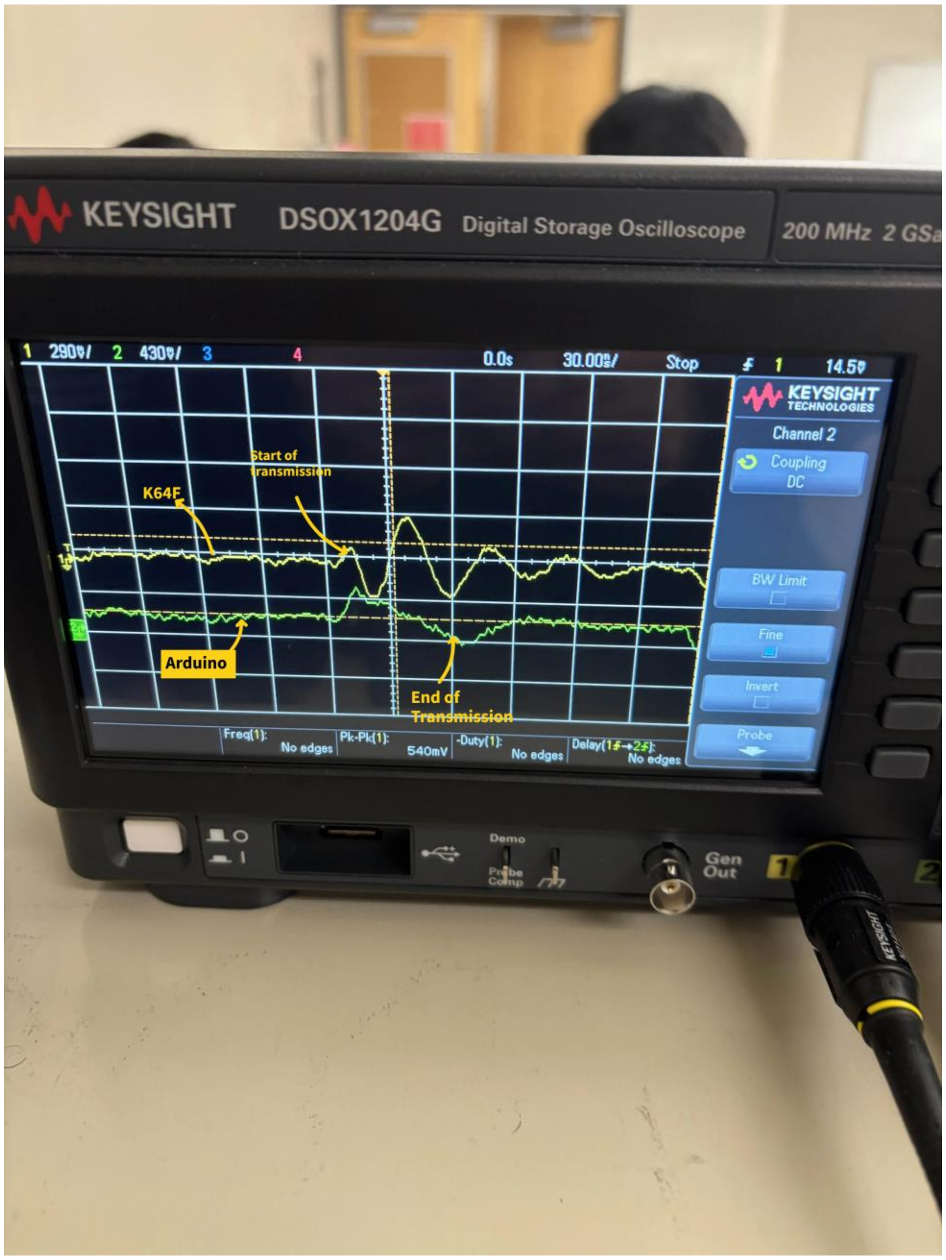
Part 2 (Arduino):

```
#include <SPI.h>
char buff [255];
volatile byte indx;
volatile boolean process;

void setup (void) {
  Serial.begin (115200);
  pinMode(MISO, OUTPUT); // have to send on master in so it set as output
  SPCR |= _BV(SPE); // turn on SPI in slave mode
  indx = 0; // buffer empty
  process = false;
  SPI.attachInterrupt(); // turn on interrupt
}

ISR (SPI_STC_vect) // SPI interrupt routine
{
  byte c = SPDR; // read byte from SPI Data Register

  if (indx < sizeof(buff)) {
    buff[indx++] = c; // save data in the next index in the array buff
    if (c == '\n') {
      buff[indx - 1] = 0; // replace newline ('\n') with end of string (0)
      process = true;
    }
  }
}

void loop (void) {
  if (process) {
    process = false; //reset the process
    Serial.println (buff); //print the array on serial monitor
    indx= 0; //reset button to zero
  }
}
```

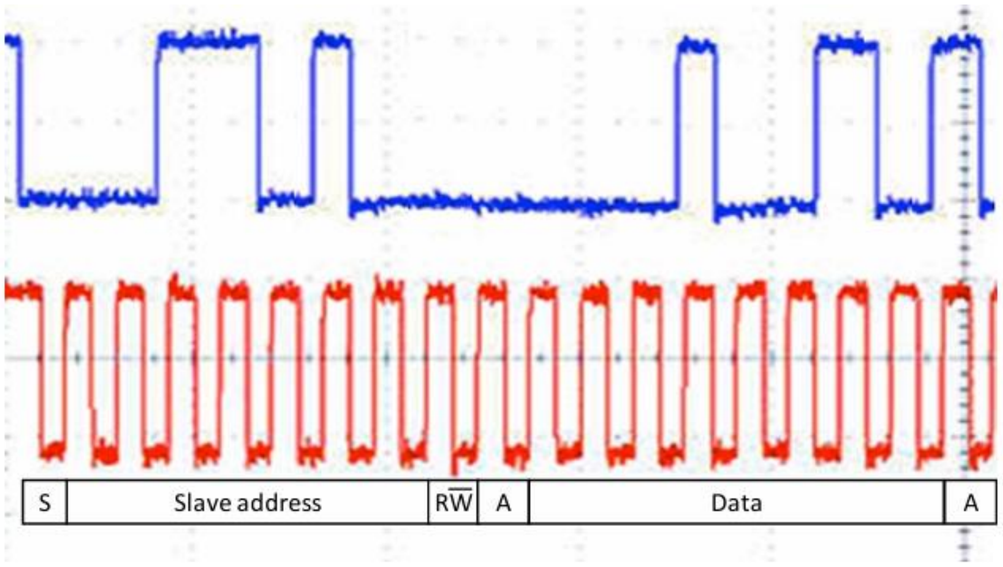
Technical Problems Encountered and Solutions

One major technical problem we encountered was setting up the oscilloscope to measure the latency of the boards. The way we eventually began to solve this was when we created a common ground between the K64F board and Arduino as well as connecting the oscilloscope to ground and power.

Questions

1. 

30 ms is the latency measure
Yes it should be reasonable as the SPI is not the one giving any delay moreso the software

2. 

A. Slave address is 0x50
B. Master is writing to slave Data = 0x30

Conclusions

In this lab, a complete communication system was built and tested with the K64F board and an Arduino Uno. The program successfully read sensor data using I2C and sent it over using SPI to the Arduino. The latency measurement worked well using GPIO pulses and viewing the timing on the oscilloscope. The data transfer across the two devices was accurate and consistent.

Team Contribution Summary

We worked together in all parts of the lab I (Muneer) worked on part of the report, and Sameer worked on another part of the report. We also both helped with debugging and verifying the oscilloscope.