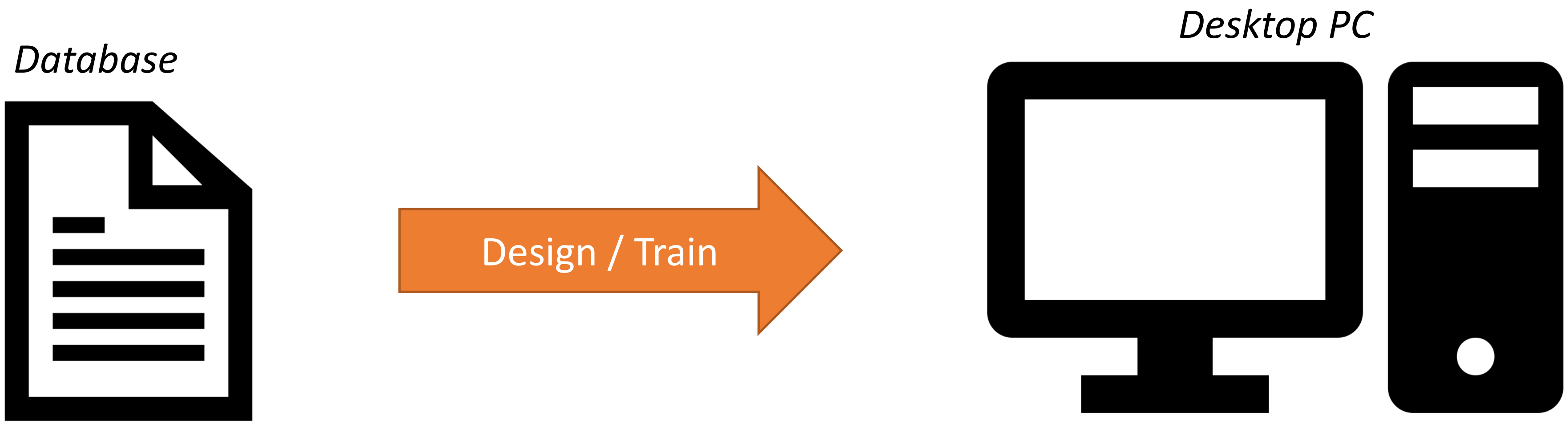
ENGR 498: Design for the Internet of Things – Hardware In the Loop Testing

Dr. Jason Forsyth
Department of Engineering
James Madison University

How can we design and test the algorithms for our embedded system?

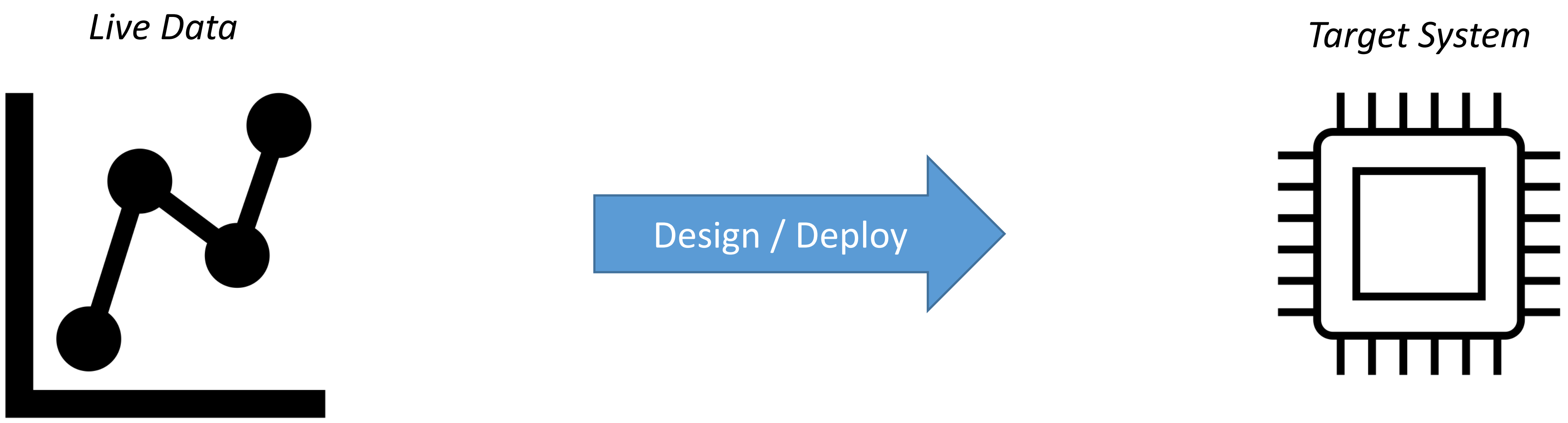
Build Our Algorithm in the Desktop with Stored Data (Offline)



- Lots of data resources
- Easier debugging and visualization tools
- Can execute quickly; lots of memory space

- Not the final system algorithm be deployed.
- May come to relay on tools that are unavailable in deploy systems
- Results/performance here may over-estimate our abilities
- Data may not map to final deploy.

Build Our Algorithm On Deployment System with Live Data (Online)

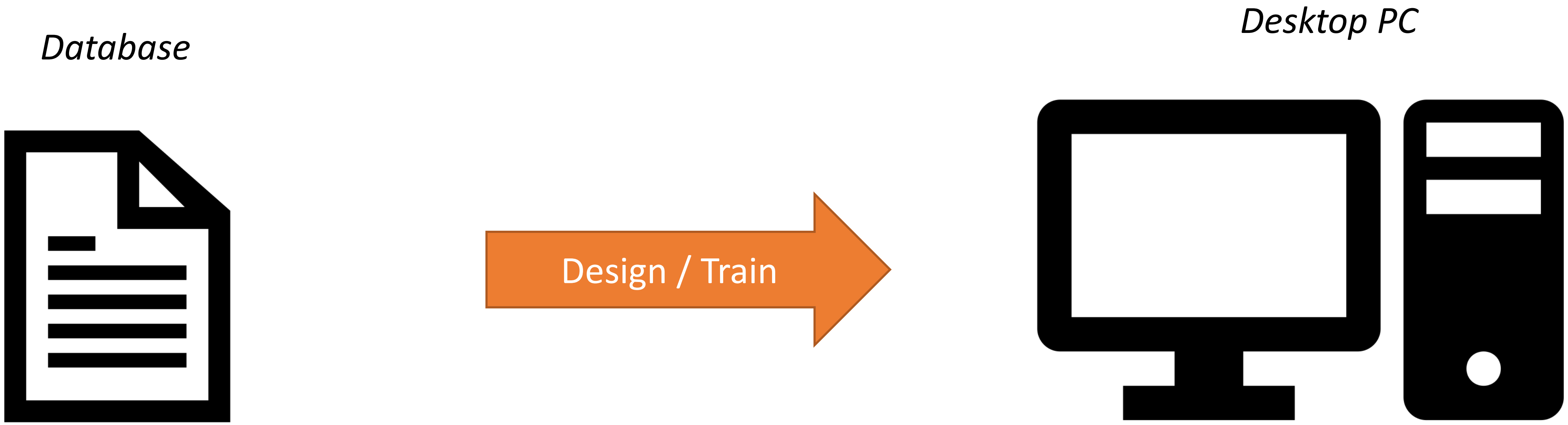


- No implementation “translation”
- Results are true for final product
- Data maps to real-world / target conditions

- Real time data is impossible to debug
- Tool chains and visualization may be limited
- May not have access to “live” data; difficult to replicate

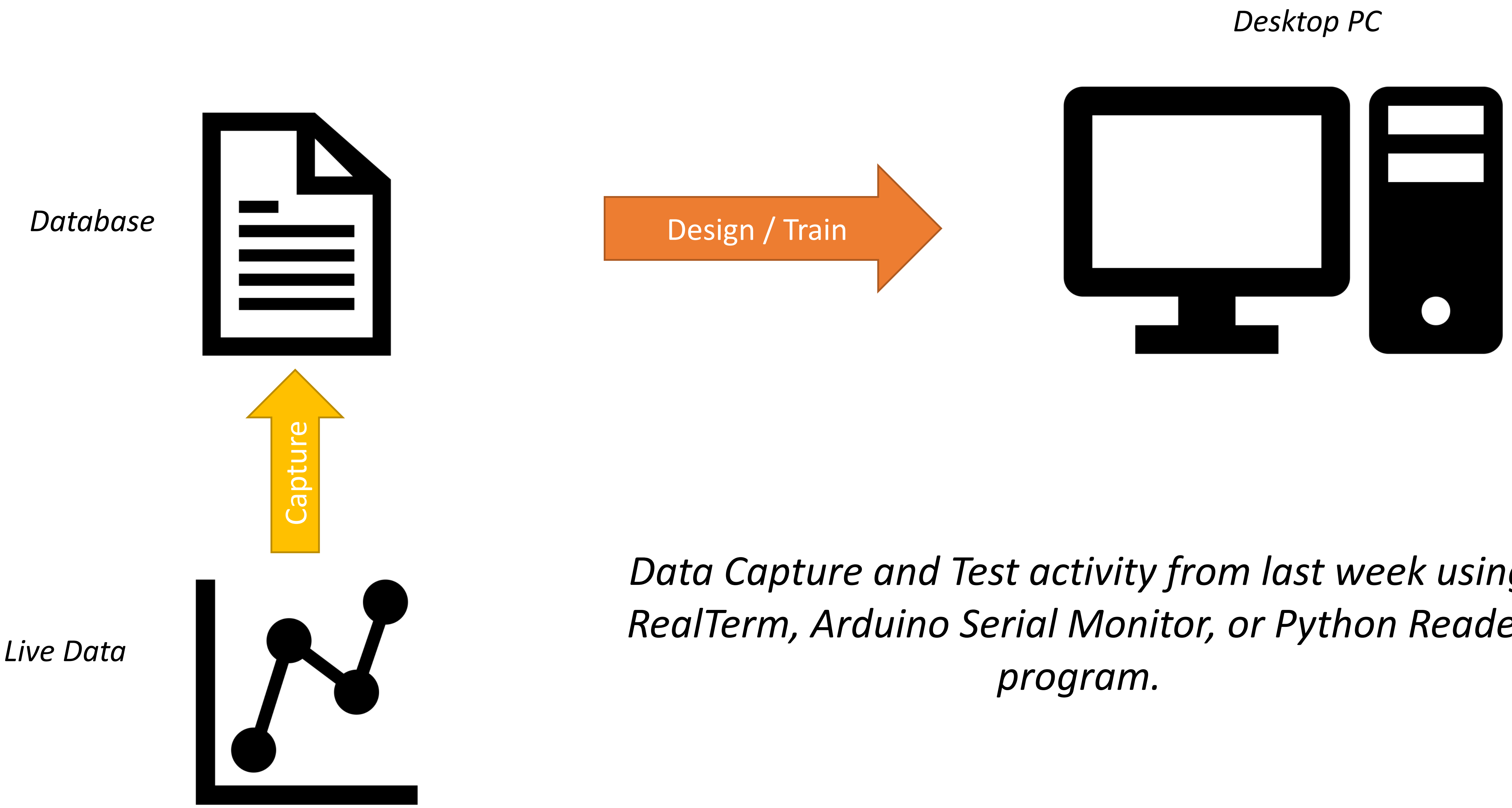
Hardware-in-the-Loop (HIL) design attempts to bridge this gap by leveraging “high-level” tools but then targeting a “low-level” implementation.

Step 1: Develop Algorithm on Desktop with Offline Data



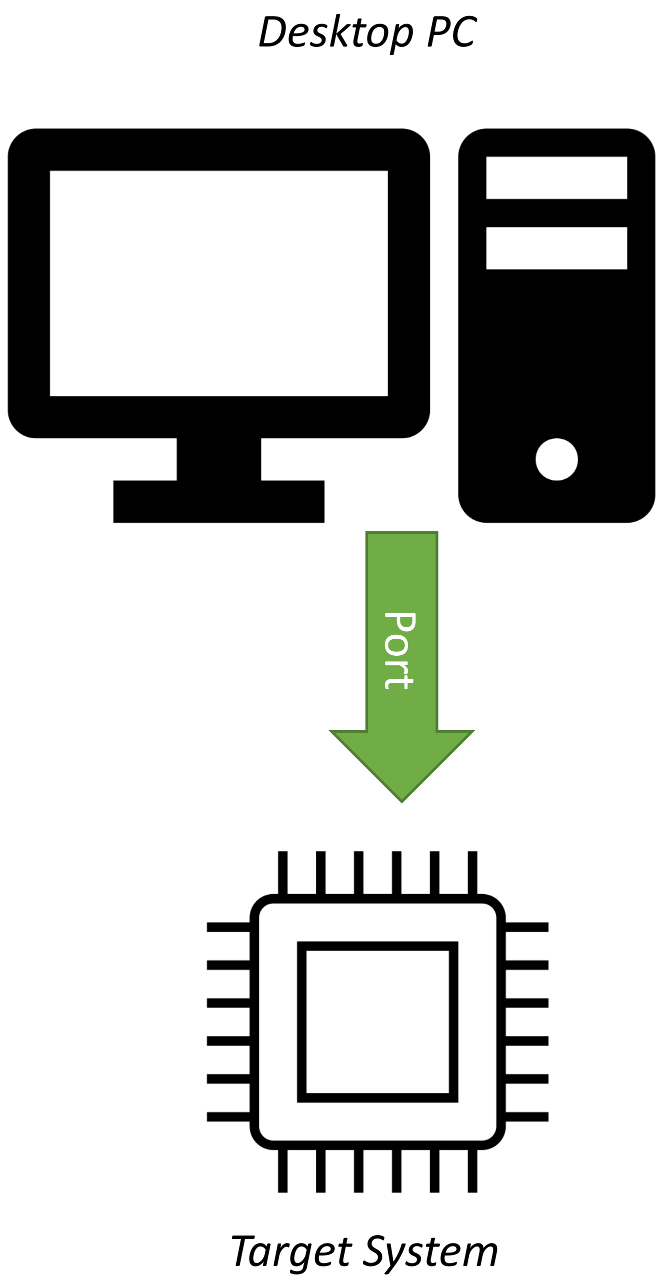
Files provided in milestone development Algorithms to Code Lecture

Step 2: Expand Offline Data through Live Capture



Data Capture and Test activity from last week using RealTerm, Arduino Serial Monitor, or Python Reader program.

Step #3: Porting Algorithm to Embedded Platform



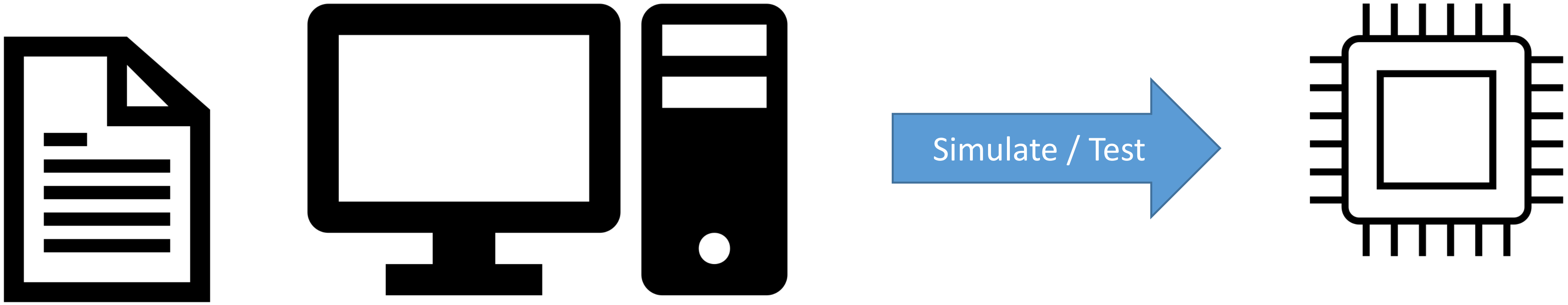
Embedded Systems Optimization Lecture discussing impact of data types and operations

Algorithms to Code lecture

Step #4: Testing the Target Algorithm

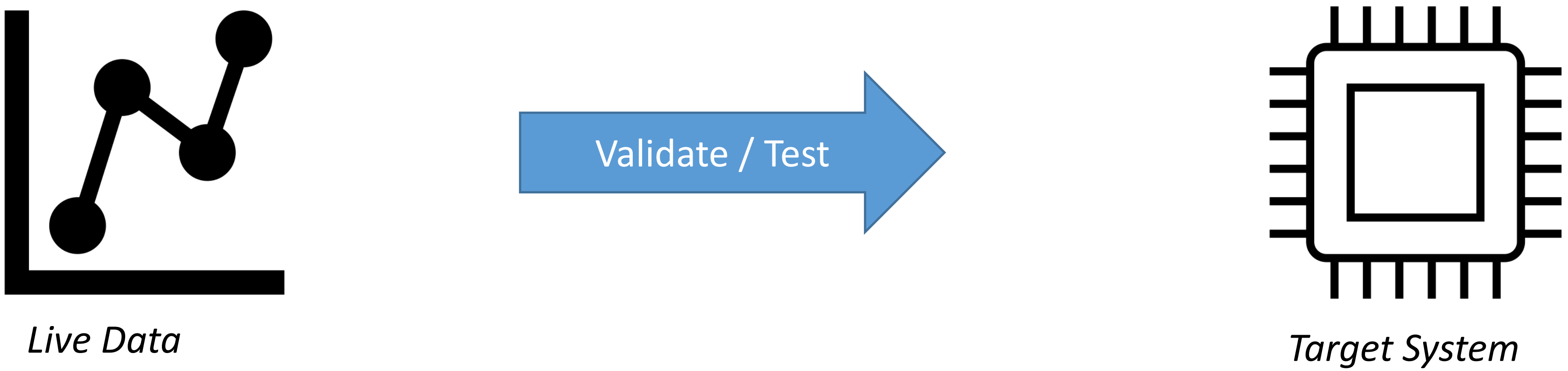
Today’s code/approach will pass files (database or capture data) from the Desktop to the target embedded system.

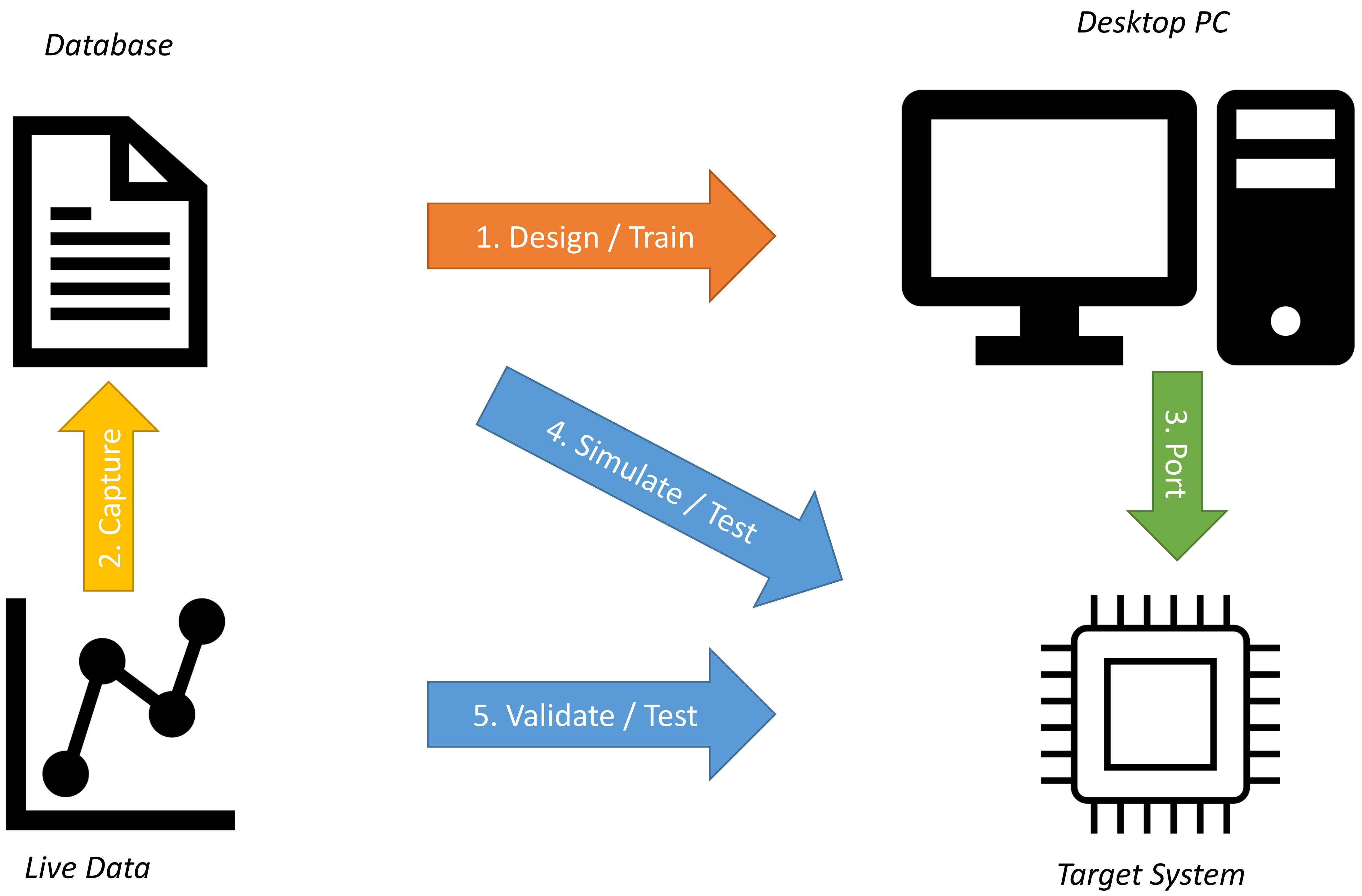
System will report results after each “step” for data passed.



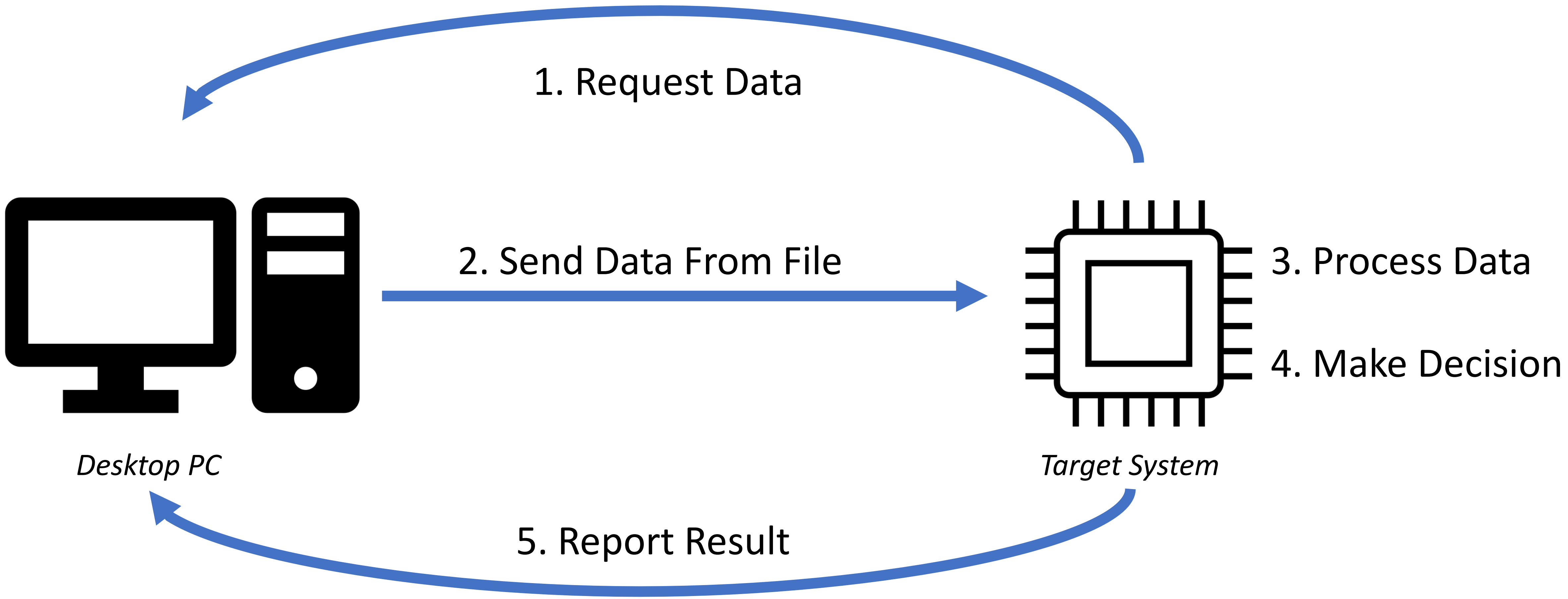
Step #5: Final Validation

Final deliverable will be demo with accelerometer and embedded system reporting live results.




Database
Desktop PC
1. Design / Train
2. Capture
3. Port
4. Simulate / Test
5. Validate / Test
Live Data
Target System

How we will implement HIL testing (offline)



```
void loop()
{
    /*****************************************************************************/
    /*  Acquire data from either the ADXL345 (online) or the Serial port (offline)
    /*****************************************************************************/
    timeAcquire=millis();
    acquire(ONLINE_MODE,xAxis,yAxis,zAxis);

    //blink an LED to ensure we're receiving data

    led_state=(led_state==HIGH)?(LOW):(HIGH);
    digitalWrite(LED_BUILTIN,led_state);

    /*****************************************************************************/
    /*  Process the data that has been acquired. This should be all the diff, squaring...etc. functions
    /*****************************************************************************/
    timeProcess=millis();
    process();

    /*****************************************************************************/
    /*  Based upon the data that has been collected, render some decision and report it back up the chain
    /*****************************************************************************/
    timeDecide=millis();
    decide();

    /*****************************************************************************/
    /*  Now that all calculations have completed; report back to the host PC
    /*****************************************************************************/
    algorithmDone=millis();
    report();


    delay(10);

}
```

Each algorithm step broken into individual functions (more on this later)

HIL client reports time to execute each function and a result value

HIL_host

```
"C:\Users\Jason Work\PycharmProjects\SerialEcho\venv\Scripts\python.exe" "C:/Users/Jason Work/PycharmProjects/SerialEcho/HIL_host.py"
Opening port COM3
Success!
Received request to Send. Sending: 1,10,111
Target reported: 10801,0,0,111
Received request to Send. Sending: 1,10,112
Target reported: 21,0,0,112
Received request to Send. Sending: 1,9,112
Target reported: 21,0,0,112
Received request to Send. Sending: 1,9,112
Target reported: 21,0,0,112
Received request to Send. Sending: 1,10,112
Target reported: 21,0,0,112
```

On PC side, each line from a data file is sent to the target.

Result is received with timing of various functions and a “result”. Which in this case is just the magnitude (more on that later)

Timing of acquire() function is currently 21ms. Was 1000ms… (more on that later)

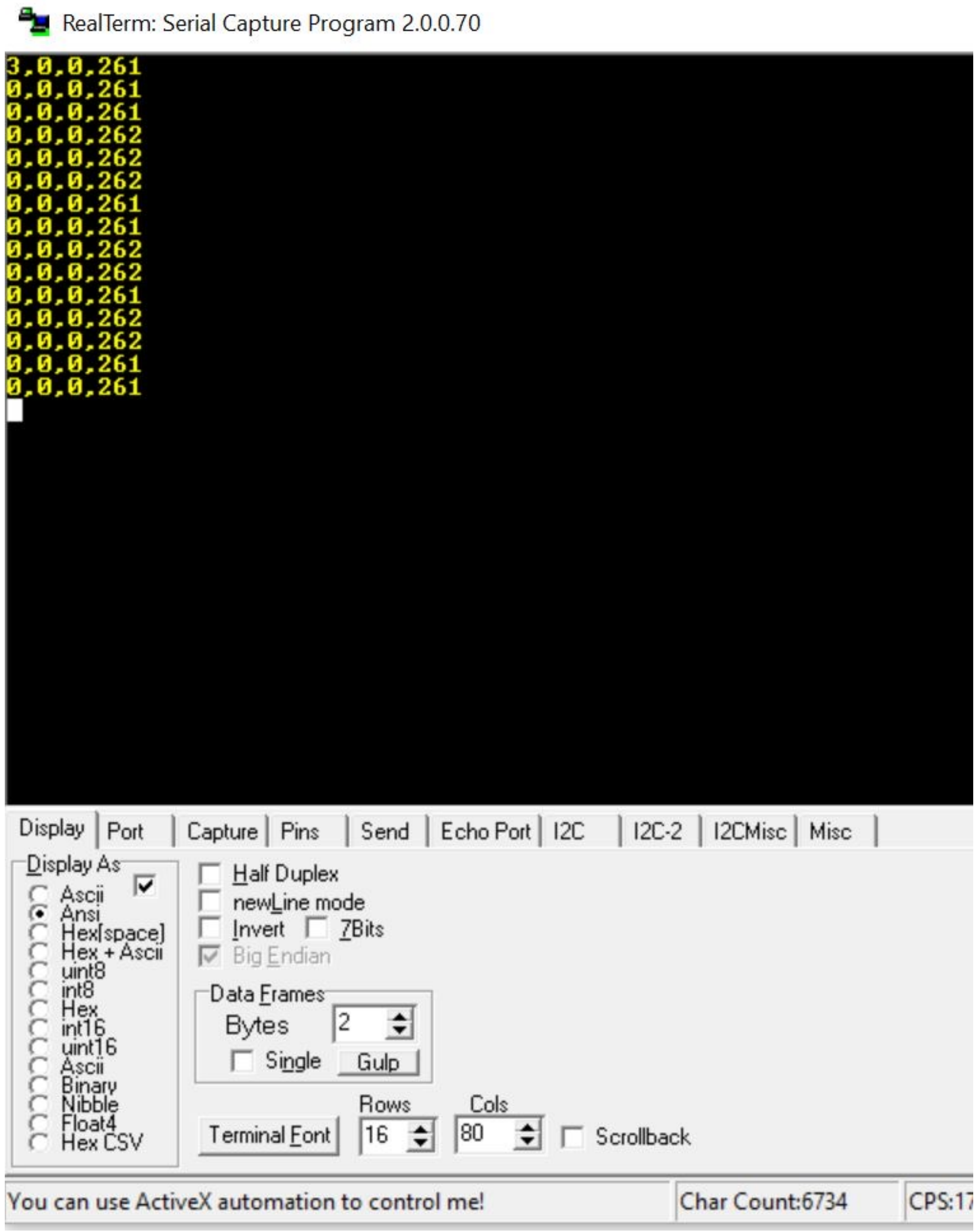
Why this whole framework and overhead…

- Can easily scan files from a directory or database and run them past the microprocessor. Might take a while but grab lunch.
- If changes/errors are noticed in target implementation, can debug them in desktop version.
- Once confident in functionality, can simply change a “flag” and target will switch from reading from files (offline) to reading from accelerometer (online).

Acquire from ADXL345

Acquire from Serial Port (PC)

```
void acquire(bool _ONLINE_ENABLE, int16_t& xAxis, int16_t& yAxis, int16_t& zAxis)
{



    //acquire data from the ADXL345
    if(_ONLINE_ENABLE==true)
    {
        xAxis = accel.getXAcceleteration();
        yAxis = accel.getYAcceleteration();
        zAxis = accel.getZAcceleteration();
    }

    //acquire data from the Serial port
    else
    {
        //wait for response
        while (Serial.available()==0)
        {
            //send '!' up the chain
            Serial.println('!');

            //spam the host until they respond :)
            delay(10);
        }

        //parse response attempt #2 (about 20ms with timeout)
        int x_accel = Serial.parseInt();
        int y_accel = Serial.parseInt();
        int z_accel = Serial.parseInt();

        xAxis=x_accel;
        yAxis=y_accel;
        zAxis=z_accel;
```


RealTerm: Serial Capture Program 2.0.0.70
3,0,0,261
0,0,0,261
0,0,0,261
0,0,0,262
0,0,0,262
0,0,0,262
0,0,0,261
0,0,0,261
0,0,0,262
0,0,0,262
0,0,0,261
0,0,0,262
0,0,0,262
0,0,0,261
0,0,0,261
Display
Port
Capture
Pins
Send
Echo Port
I2C
I2C-2
I2CMisc
Misc
Display As
Ascii
Ansi
Hex[space]
Hex + Ascii
uint8
int8
Hex
int16
uint16
Ascii
Binary
Nibble
Float4
Hex CSV
Half Duplex
newLine mode
Invert
7Bits
Big Endian
Data Frames
Bytes
2
Single
Gulp
Terminal Font
Rows
16
Cols
80
Scrollback
You can use ActiveX automation to control me!
Char Count:6734
CPS:17

Can report “anything” back to the PC that would be useful for debugging.

```
void report()
{
    //calculate how long the acquire() took
    long time_to_acquire = timeProcess - timeAcquire;

    //calculate how long the processs() took
    long time_to_process = timeProcess - timeDecide;

    //calculate how long the decide() took
    long time_to_decide = timeDecide - algorithmDone;

    //print all back to host PC
    Serial.print(time_to_acquire);
    Serial.print(",");
    Serial.print(time_to_process);
    Serial.print(",");
    Serial.print(time_to_decide);
    Serial.print(",");
    Serial.println(magnitude); //print back the magnitude for fun so we can see if the calculation was done correctly.

}
```

Now for some "fun" things I learned building this out…

```
void loop() {

    //read x-axis
    volatile int16_t xAxis=accel.getXAcceleteration();

    //read y-axis
    volatile int16_t yAxis=accel.getYAcceleteration();

    //read z-axis
    volatile int16_t zAxis=accel.getZAcceleteration();

    //your Arduino will hate this but we need to test....
    volatile int16_t magnitude = sqrt(xAxis*xAxis + yAxis*yAxis + zAxis*zAxis);

    //result should be ~1g when ADXL345 is not moving
    float acceleration = accel.convertRawToFloat(magnitude);
}
```

100 %

Locals

Name	Value
xAxis	12
yAxis	11
zAxis	261
magnitude	53

???

$$\sqrt{12^2 + 11^2 + 261^2} = 261$$

```
void loop() {

    //read x-axis
    volatile long xAxis=accel.getXAcceleteration();

    //read y-axis
    volatile long yAxis=accel.getYAcceleteration();

    //read z-axis
    volatile long zAxis=accel.getZAcceleteration();

    //your Arduino will hate this but we need to test....(actually on the 32u4 i
    volatile int16_t magnitude = sqrt(xAxis*xAxis + yAxis*yAxis + zAxis*zAxis);

    //result should be ~1g when ADXL345 is not moving
    volatile float acceleration = accel.convertRawToFloat(magnitude);
```

Name	Value
xAxis	17
yAxis	14
zAxis	264
magnitude	264
acceleration	1.03125
num	11269

$$\sqrt{17^2 + 14^2 + 264^2} = 264$$

What’s going on?

An *int16_t* can’t hold the result of the multiply operation. Must use a *long* the individual data types.

Originally this code took 1000ms to execute

Other String parsing implementations were attempted (including the one last week). No impact.

Baud rate changed from 9600 bps to 115200 bps. No impact.

println() is very quick. We know this from when the serial monitor is spammed.

```
//wait for response
while (Serial.available()==0)
{
    //send '!' up the chain
    Serial.println('!');

    //spam the host until they respond :)
    delay(10);
}
```

```
//parse response attempt #2 (about 20ms with timeout)
int x_accel = Serial.parseInt();
int y_accel = Serial.parseInt();
int z_accel = Serial.parseInt();

xAxis=x_accel;
yAxis=y_accel;
zAxis=z_accel;
```

Left code ran instantaneously….

```
//wait for response
while (Serial.available()==0)
{
    //send '!' up the chain
    Serial.println('!');

    //spam the host until they respond :)
    delay(10);
}

while(Serial.available>0){Serial.read();}
```

So which function is the problem? And what do we do about it?

```
//wait for response
while (Serial.available()==0)
{
    //send '!' up the chain
    Serial.println('!');

    //spam the host until they res
    delay(10);
}
```

```
//parse response attempt #2 (abou
int x_accel = Serial.parseInt();
int y_accel = Serial.parseInt();
int z_accel = Serial.parseInt();

xAxis=x_accel;
yAxis=y_accel;
zAxis=z_accel;
```

RTFM....place Serial.setTimeout() in setup()

Serial.parseInt()

Description

Looks for the next valid integer in the incoming serial. The function terminates if it times out (see Serial.setTimeout()).

Serial.setTimeout()

Description

`Serial.setTimeout()` sets the maximum milliseconds to wait for serial data. It defaults to 1000 milliseconds.

Putting this all together…

Performing Hardware-in-the-Loop Testing

- HIL Host, a Python program, has been provided (link). *Yes, I'm asking you to install yet-another-programming-language (YAPL) but it will make validation easier.*
- HIL Client, an Atmel/Arduino sketch implementing this framework (link).
- My ADXL345 class (.h, .cpp).
- Milestone: Have this working on offline and online data by December 2nd.

Getting Started on HIL Host

- Download PyCharm and read docs: https://www.jetbrains.com/help/pycharm/quick-start-guide.html
- Make a new PyCharm / Python project. Copy and paste my code inside.
- Install serial package (right click on top of file and select “install dependencies)”
- Change *portName* line to match your computer.
- Copy .csv file into same location and HIL_host.py program.
- Press Green button to GO!

Getting Started on HIL Client

- Make a new Atmel project from an Arduino sketch.
- Copy and paste my code into Sketch.cpp that is generated.
- Create new ADXL345.h and ADXL345.cpp files. Add your code or mine.
- Step delay is set to 100ms in main(). Once system is up and running can reduce to 5-10ms.

With Both Host and Client Installed

- In Atmel, *Debug -> Start Without Debugging* to begin running HIL Client. This must be done first.
- In PyCharm, press Green button to GO!
- Read results from PyCharm terminal.
- If desire “online” mode. Change above loop() and read results of live data from RealTerm or Arduino Serial Monitor.

Thanks for hanging in there with me ☺