Arduino IDE
I2C

Jaehyun Lee Shawn Hagler

Arduino IDE

- **Arduino**
  - “Open-source electronic prototyping platform enabling users to create interactive electronic objects.”

- **IDE**
  - An Integrated Development Environment
    - write, validate, compile, upload, verify and then interact with programmed board
- Validating and Compiling code
  - Arduino IDE uses installed compiler defined by board settings
    - GNU-GCC compiler to validate and convert C code
- Uploading and Verifying code
  - Arduino IDE calls corresponding tool based on board
    - Avrdude tool: writes compiled code to microcontroller
      allows access to flash and EEPROM of the chip, among other functions
- Interacting with the microcontrollers
  - Arduino IDE uses corresponding SPI, I2C, or UART tool based on board

Arduino Programming

```
1  void setup() {
2    // put your setup code here, to run once:
3
4  }
5
6  void loop() {
7    // put your main code here, to run repeatedly:
8
9  }
```

- The setup function
  - To initialize input and output pins so they are ready to be used
- The loop function
  - The program calls the code inside the loop function repeatedly until the Arduino board is powered off

Arduino Programming cont.

```
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  }
```

- The setup function
  - To initialize input and output pins so they are ready to be used
- The loop function
  - The program calls the code inside the loop function repeatedly until the Arduino board is powered off

- The main function located in the main.cpp file in the core directory of Arduino program packages
- The main function consistently checks for the serial event

Arduino Programming cont.

```
64  void serialEventRun(void)
65  {
66  #if defined(HAVE_HWSERIAL0)
67    if (Serial0_available && serialEvent && Serial0_available()) serialEvent();
68  #endif
69  #if defined(HAVE_HWSERIAL1)
70    if (Serial1_available && serialEvent1 && Serial1_available()) serialEvent1();
71  #endif
72  #if defined(HAVE_HWSERIAL2)
73    if (Serial2_available && serialEvent2 && Serial2_available()) serialEvent2();
74  #endif
75  #if defined(HAVE_HWSERIAL3)
76    if (Serial3_available && serialEvent3 && Serial3_available()) serialEvent3();
77  #endif
78  }
```

- The serialEventRun function located in the HardwareSerial.cpp
- SerialEvent() is called at the end of loop() when data is available
  - Can use Serial.read() to capture the data

- The main function located in the main.cpp file in the core directory of Arduino program packages
- The main function consistently checks for the serial event

```
73  // Function that can be weakly referenced by serialEventRun to prevent
74  // pulling in this file if it's not otherwise used.
75  bool Serial0_available() {
76    return Serial.available();
77  }
```

Wire Library

```
class TwoWire : public Stream
{
  private:
    static uint8_t rxBuffer[];
    static uint8_t rxBufferIndex;
    static uint8_t rxBufferLength;

    static uint8_t txAddress;
    static uint8_t txBuffer[];
    static uint8_t txBufferIndex;
    static uint8_t txBufferLength;

    static uint8_t transmitting;
    static void (*user_onRequest)(void);
    static void (*user_onReceive)(int);
    static void onRequestService(void);
    static void onReceiveService(uint8_t*, int);
```

- Receive and Transmit Buffers
- write() and read()
- onReceive()

```
  public:
    TwoWire();
    void begin();
    void begin(uint8_t);
    void begin(int);
    void end();
    void setClock(uint32_t);
    void setWireTimeout(uint32_t timeout = 25000, bool reset_with_timeout = false);
    bool getWireTimeoutFlag(void);
    void clearWireTimeoutFlag(void);
    void beginTransmission(uint8_t);
    void beginTransmission(int);
    uint8_t endTransmission(void);
    uint8_t endTransmission(uint8_t);
    uint8_t requestFrom(uint8_t, uint8_t);
    uint8_t requestFrom(uint8_t, uint8_t, uint8_t);
    uint8_t requestFrom(uint8_t, uint8_t, uint32_t, uint8_t, uint8_t);
    uint8_t requestFrom(int, int);
    uint8_t requestFrom(int, int, int);
    virtual size_t write(uint8_t);
    virtual size_t write(const uint8_t *, size_t);
    virtual int available(void);
    virtual int read(void);
    virtual int peek(void);
    virtual void flush(void);
    void onReceive( void (*)(int) );
    void onRequest( void (*)(void) );

    inline size_t write(unsigned long n) { return write((uint8_t)n); }
    inline size_t write(long n) { return write((uint8_t)n); }
    inline size_t write(unsigned int n) { return write((uint8_t)n); }
    inline size_t write(int n) { return write((uint8_t)n); }
    using Print::write;
};

extern TwoWire Wire;

#endif
```

Wire Library cont.

- write()
  - Calls a twi_transmit function with the parameters of `const uint8_t* data, uint8_t length'
  - The data is then write to the tx buffer

```
// must be called in:
// slave tx event callback
// or after beginTransmission(address)
size_t TwoWire::write(const uint8_t *data, size_t quantity)
{
  if(transmitting){
  // in master transmitter mode
    for(size_t i = 0; i < quantity; ++i){
      write(data[i]);
    }
  }else{
  // in slave send mode
    // reply to master
    twi_transmit(data, quantity);
  }
  return quantity;
}
```

```
/*
 * Function twi_transmit
 * Desc     fills slave tx buffer with data
 *          must be called in slave tx event callback
 * Input    data: pointer to byte array
 *          length: number of bytes in array
 * Output   1 length too long for buffer
 *          2 not slave transmitter
 *          0 ok
 */
uint8_t twi_transmit(const uint8_t* data, uint8_t length)
{
  uint8_t i;

  // ensure data will fit into buffer
  if(TWI_BUFFER_LENGTH < (twi_txBufferLength+length)){
    return 1;
  }

  // ensure we are currently a slave transmitter
  if(TWI_STX != twi_state){
    return 2;
  }

  // set length and copy data into tx buffer
  for(i = 0; i < length; ++i){
    twi_txBuffer[twi_txBufferLength+i] = data[i];
  }
  twi_txBufferLength += length;

  return 0;
}
```

Wire Library cont.

- read()

```
288  // must be called in:
289  // slave rx event callback
290  // or after requestFrom(address, numBytes)
291  int TwoWire::read(void)
292  {
293    int value = -1;
294
295    // get each successive byte on each call
296    if(rxBufferIndex < rxBufferLength){
297      value = rxBuffer[rxBufferIndex];
298      ++rxBufferIndex;
299    }
300
301    return value;
302  }
```

Wire Library cont.

- onReceive()
  - sets the on receive event to the given function `(void (*function)(uint8_t*, size_t))` `

```
363  // sets function called on slave write
364  void TwoWire::onReceive( void (*function)(int) )
365  {
366    user_onReceive = function;
367  }
```

```
323  // behind the scenes function that is called when data is received
324  void TwoWire::onReceiveService(uint8_t* inBytes, int numBytes)
325  {
326    // don't bother if user hasn't registered a callback
327    if(!user_onReceive){
328      return;
329    }
330    // don't bother if rx buffer is in use by a master requestFrom() op
331    // i know this drops data, but it allows for slight stupidity
332    // meaning, they may not have read all the master requestFrom() data yet
333    if(rxBufferIndex < rxBufferLength){
334      return;
335    }
336    // copy twi rx buffer into local read buffer
337    // this enables new reads to happen in parallel
338    for(uint8_t i = 0; i < numBytes; ++i){
339      rxBuffer[i] = inBytes[i];
340    }
341    // set rx iterator vars
342    rxBufferIndex = 0;
343    rxBufferLength = numBytes;
344    // alert user program
345    user_onReceive(numBytes);
346  }
```

I2C Programming

- Master Code structure

```
/* master code */
void setup() {
  // initiate the Wire library
  Wire.begin()
}

void loop() {
  // begin transmission to the slave
  // sets the tx address to the value passed
  // turns on a flag that states we are transmitting
  Wire.beginTransmission(9);

  // transmit a value over the transmission buffer
  // all write commands are batched together
  Wire.write(x);

  // end transmission to the slave
  // actually writes to the slave buffer
  Wire.endTransmission();
}
```

- Master Code example

```
1  #include Wire.h
2
3  int x = 0;
4  void setup() {
5    // Start the I2C Bus as Master
6    Wire.begin();
7  }
8  void loop() {
9    Wire.beginTransmission(9); // transmit to device #9
10   Wire.write(x);              // sends x
11   Wire.endTransmission();    // stop transmitting
12   x++; // Increment x
13   if (x > 5) x = 0; // `reset x once it gets 6
14   delay(500);
15 }
```

I2C Programming cont.

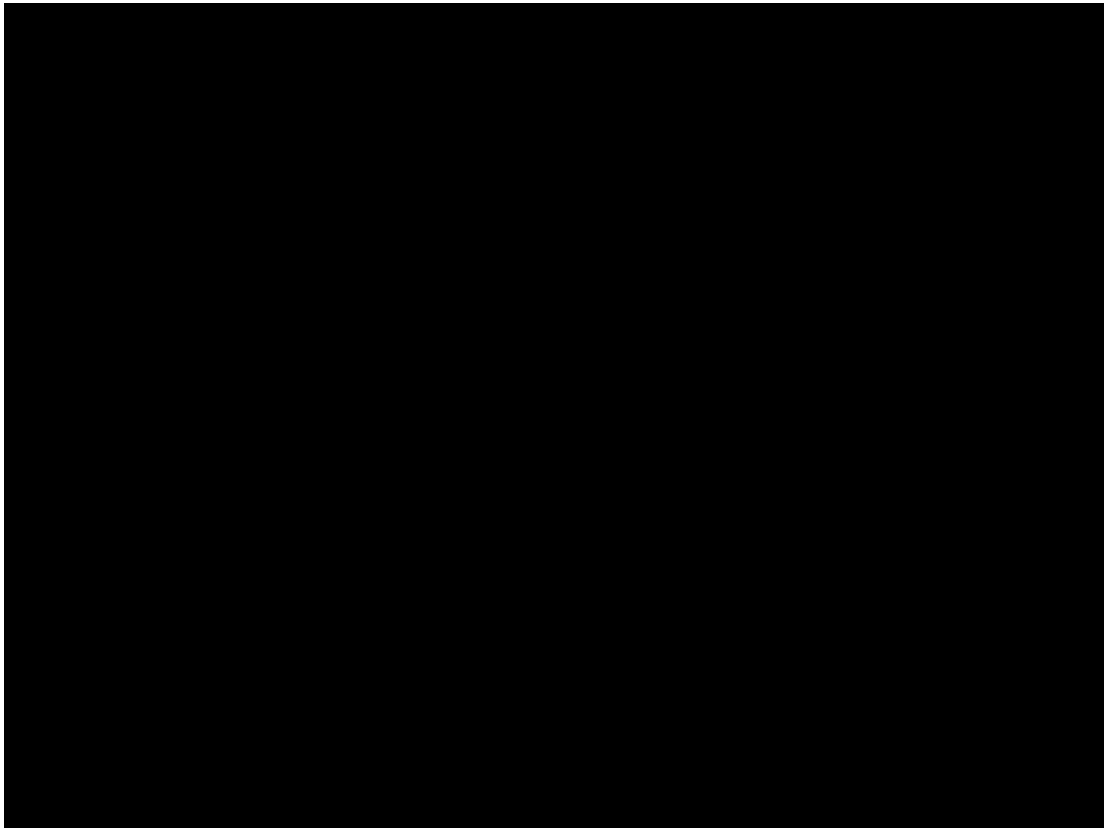
- Slave Code structure

```
/* slave code */
void setup() {
  // initiate the Wire library
  // starts the I2C bus as slave on the passed tx address
  Wire.begin(9);

  // attaches a function callback to the recieve event
  // this function is called when ever the I2C bus notices the buffer changes
  Wire.onRecieve(receiveEvent);
}

void receiveEvent(int bytes) {
  // reads one character from the I2C bus
  x = Wire.read();
}
```

- Slave Code example

```
#include Wire.h

int LED = 13;
int x = 0;
void setup() {
  // Define the LED pin as Output
  pinMode (LED, OUTPUT);
  // Start the I2C Bus as Slave on address 9
  Wire.begin(9);
  // Attach a function to trigger when something is received.
  Wire.onReceive(receiveEvent);
}
void receiveEvent(int bytes) {
  x = Wire.read();    // read one character from the I2C
}
void loop() {
  //If value received is 0 blink LED for 200 ms
  if (x == '0') {
    digitalWrite(LED, HIGH);
    delay(200);
    digitalWrite(LED, LOW);
    delay(200);
  }
  //If value received is 3 blink LED for 400 ms
  if (x == '3') {
    digitalWrite(LED, HIGH);
    delay(400);
    digitalWrite(LED, LOW);
    delay(400);
  }
}
```

Demo



Thank You!