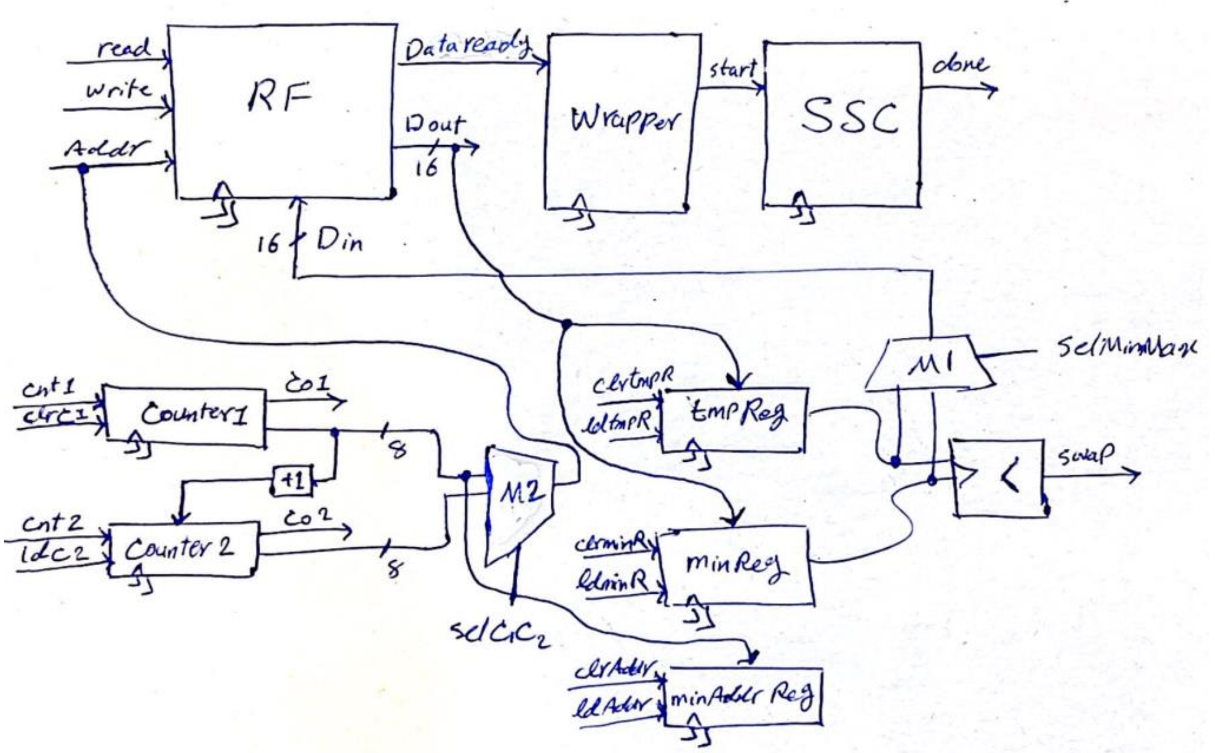
University of Tehran
Electrical and Computer Engineering Department
ECE (8101) 342
Object Oriented Modeling of Electronic Circuits – Spring 1402-03
Computer Assignment 4
Zainalabedin Navabi

Name: Amirhesam Jafari rad

SID: 810100247

First of all lets have a look at the datapath ,in which three states, namely update0, update1, and update2, are assessed according to the CA deliveries.

DataPath:

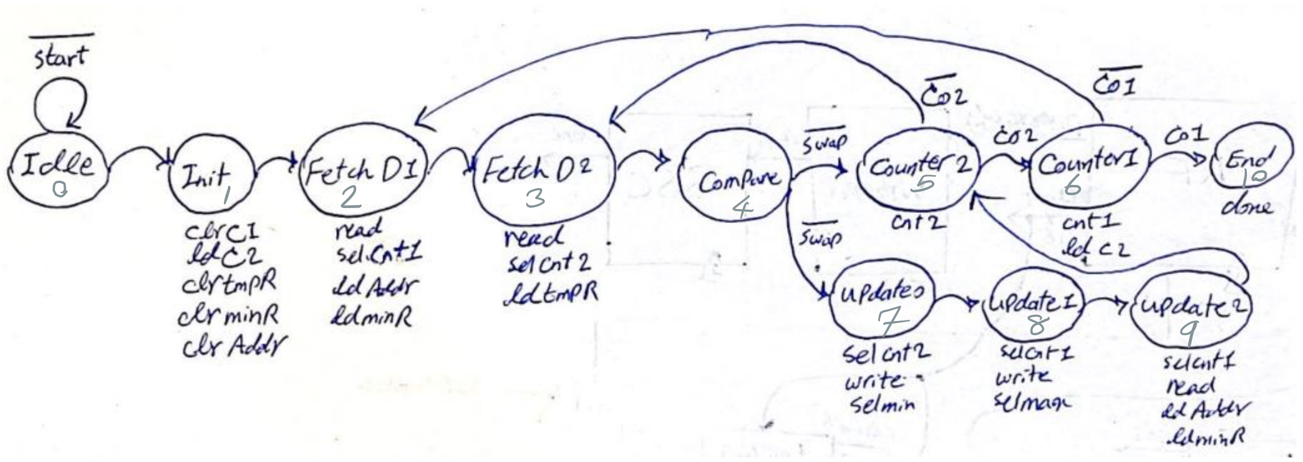


Note that I've only implemented the SSC Unit which contains register file, and not wrapper. As shown, there are three top modules named RF, Wrapper, and SSC corresponding to the register file, processing element and Selection Sorter Circuit respectively.

Then, all needed components for SSC is prepared, consisting of 2 counters, 2 muxes, 3 registers and a comparator. The +1 unit that connect Counter1 output to the Counter1 input isn't implemented as an independent but is assumed to be done internally.

The controller of the SSC design is as below:

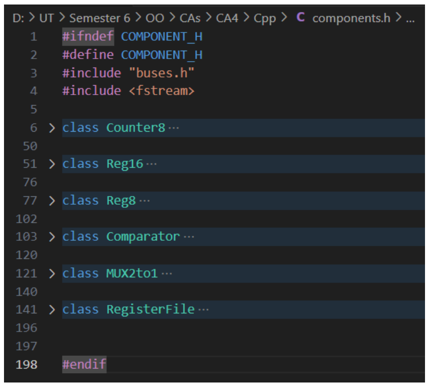
Controller:



As we can see, there are separated states for each operation. However, in reality, the Compare state was not that much necessary. In CPP implementation, the absence of this state leads to incorrect result due to incomplete inputs of the comparator unit.

CPP Implementation:

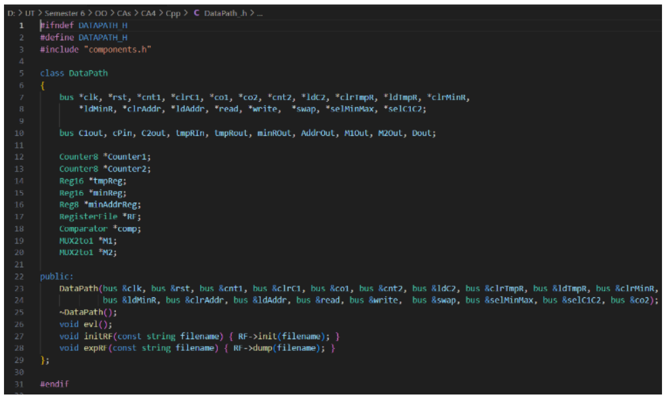
The library I've used for my wiring was buses.h which was prepared earlier. Then I create all my modules with their functionality implicitly in a components.h (Fig. 1) file where the components are defined as a C++ class. Then I instatiated each module in DataPath_.h file which consist of instantiated modules and aa evl function, in which the whole datapath is evaluated. The content of evl method is declared in DataPath_.cpp file (Fig. 2,3). After implementing datapath, I created a Controller.h file in which the states of shown controller is defined. It also has a .cpp file in which same as datapath, evaluations are handled (Fig. 4,5).
Finally I've write a top module file namely SSC.h and SSC.cpp which only instantiate the datapath and controller and evaluate them sequentially. The final file to test the design named testBench.cpp with the proper test cases. We can see that a memory named "mem.txt" is sorted in descending order in an "out.txt" file (Fig. 6,7).



D: > UT > Semester 6 > OO > CAs > CA4 > Cpp > components.h > ...

```cpp
#ifndef COMPONENT_H
#define COMPONENT_H
#include "buses.h"
#include <fstream>

class Counter8 ···

class Reg16 ···

class Reg8 ···

class Comparator ···

class MUX2to1 ···

class RegisterFile ···


#endif
```

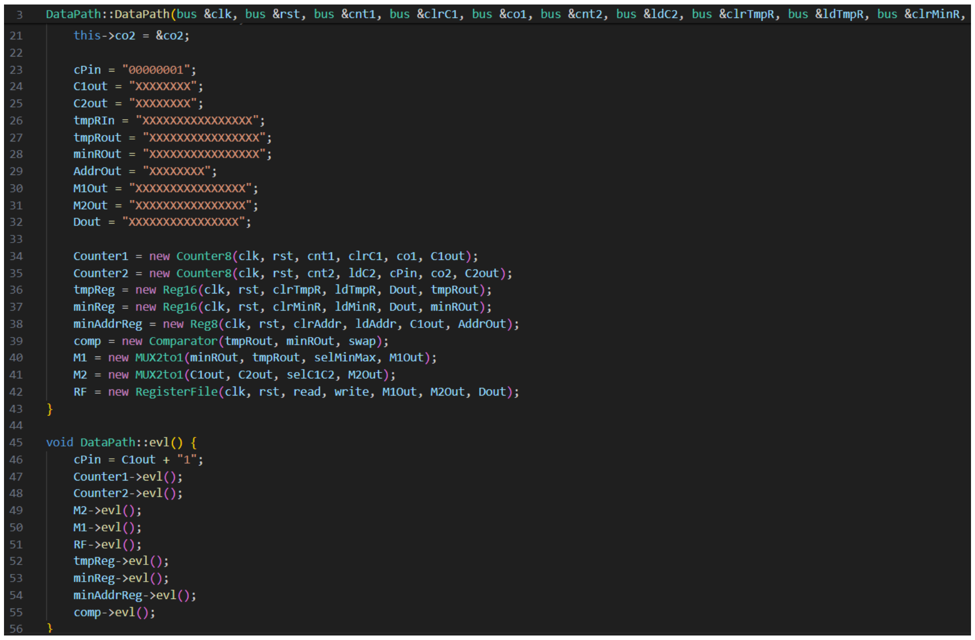
Fig. 1



D: > UT > Semester 6 > OO > CAs > CA4 > Cpp > DataPath_.h > ...

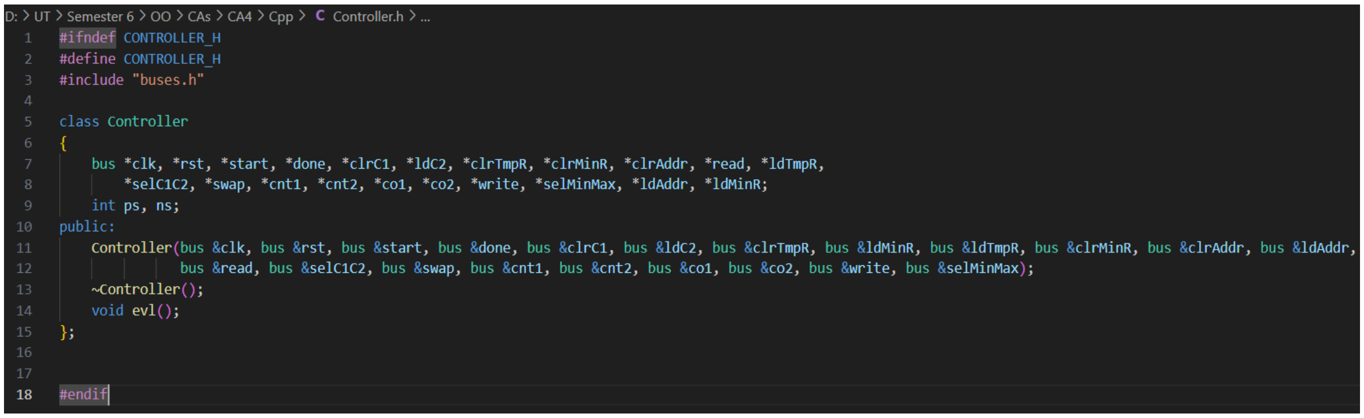
```cpp
#ifndef DATAPATH_H
#define DATAPATH_H
#include "components.h"

class DataPath
{
    bus *clk, *rst, *cnt1, *clrC1, *co1, *co2, *cnt2, *ldC2, *clrTmpR, *ldTmpR, *clrMinR,
        *ldMinR, *clrAddr, *ldAddr, *read, *write,  *swap, *selMinMax, *selC1C2;

    bus C1out, cPin, C2out, tmpRIn, tmpRout, minROut, AddrOut, M1Out, M2Out, Dout;

    Counter8 *Counter1;
    Counter8 *Counter2;
    Reg16 *tmpReg;
    Reg16 *minReg;
    Reg8 *minAddrReg;
    RegisterFile *RF;
    Comparator *comp;
    MUX2to1 *M1;
    MUX2to1 *M2;

public:
    DataPath(bus &clk, bus &rst, bus &cnt1, bus &clrC1, bus &co1, bus &cnt2, bus &ldC2, bus &clrTmpR, bus &ldTmpR, bus &clrMinR,
             bus &ldMinR, bus &clrAddr, bus &ldAddr, bus &read, bus &write,  bus &swap, bus &selMinMax, bus &selC1C2, bus &co2);
    ~DataPath();
    void evl();
    void initRF(const string filename) { RF->init(filename); }
    void expRF(const string filename) { RF->dump(filename); }
};

#endif
```

Fig. 2



```cpp
DataPath::DataPath(bus &clk, bus &rst, bus &cnt1, bus &clrC1, bus &co1, bus &cnt2, bus &ldC2, bus &clrTmpR, bus &ldTmpR, bus &clrMinR,
    this->co2 = &co2;

    cPin = "00000001";
    C1out = "XXXXXXXX";
    C2out = "XXXXXXXX";
    tmpRIn = "XXXXXXXXXXXXXXXX";
    tmpRout = "XXXXXXXXXXXXXXXX";
    minROut = "XXXXXXXXXXXXXXXX";
    AddrOut = "XXXXXXXX";
    M1Out = "XXXXXXXXXXXXXXXX";
    M2Out = "XXXXXXXXXXXXXXXX";
    Dout = "XXXXXXXXXXXXXXXX";

    Counter1 = new Counter8(clk, rst, cnt1, clrC1, co1, C1out);
    Counter2 = new Counter8(clk, rst, cnt2, ldC2, cPin, co2, C2out);
    tmpReg = new Reg16(clk, rst, clrTmpR, ldTmpR, Dout, tmpRout);
    minReg = new Reg16(clk, rst, clrMinR, ldMinR, Dout, minROut);
    minAddrReg = new Reg8(clk, rst, clrAddr, ldAddr, C1out, AddrOut);
    comp = new Comparator(tmpRout, minROut, swap);
    M1 = new MUX2to1(minROut, tmpRout, selMinMax, M1Out);
    M2 = new MUX2to1(C1out, C2out, selC1C2, M2Out);
    RF = new RegisterFile(clk, rst, read, write, M1Out, M2Out, Dout);
}

void DataPath::evl() {
    cPin = C1out + "1";
    Counter1->evl();
    Counter2->evl();
    M2->evl();
    M1->evl();
    RF->evl();
    tmpReg->evl();
    minReg->evl();
    minAddrReg->evl();
    comp->evl();
}
```

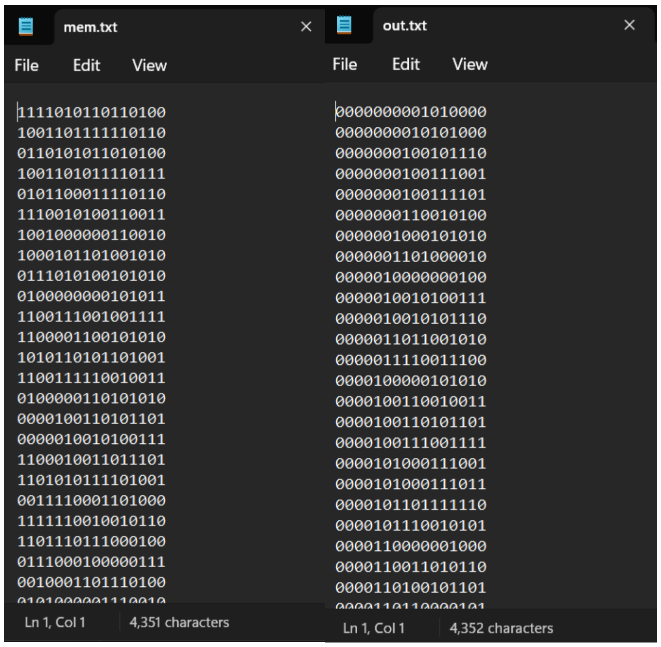
Fig. 3



D: > UT > Semester 6 > OO > CAs > CA4 > Cpp > Controller.h > ...

```cpp
#ifndef CONTROLLER_H
#define CONTROLLER_H
#include "buses.h"

class Controller
{
    bus *clk, *rst, *start, *done, *clrC1, *ldC2, *clrTmpR, *clrMinR, *clrAddr, *read, *ldTmpR,
        *selC1C2, *swap, *cnt1, *cnt2, *co1, *co2, *write, *selMinMax, *ldAddr, *ldMinR;
    int ps, ns;
public:
    Controller(bus &clk, bus &rst, bus &start, bus &done, bus &clrC1, bus &ldC2, bus &clrTmpR, bus &ldMinR, bus &ldTmpR, bus &clrMinR, bus &clrAddr, bus &ldAddr,
               bus &read, bus &selC1C2, bus &swap, bus &cnt1, bus &cnt2, bus &co1, bus &co2, bus &write, bus &selMinMax);
    ~Controller();
    void evl();
};


#endif
```

Fig. 4,5



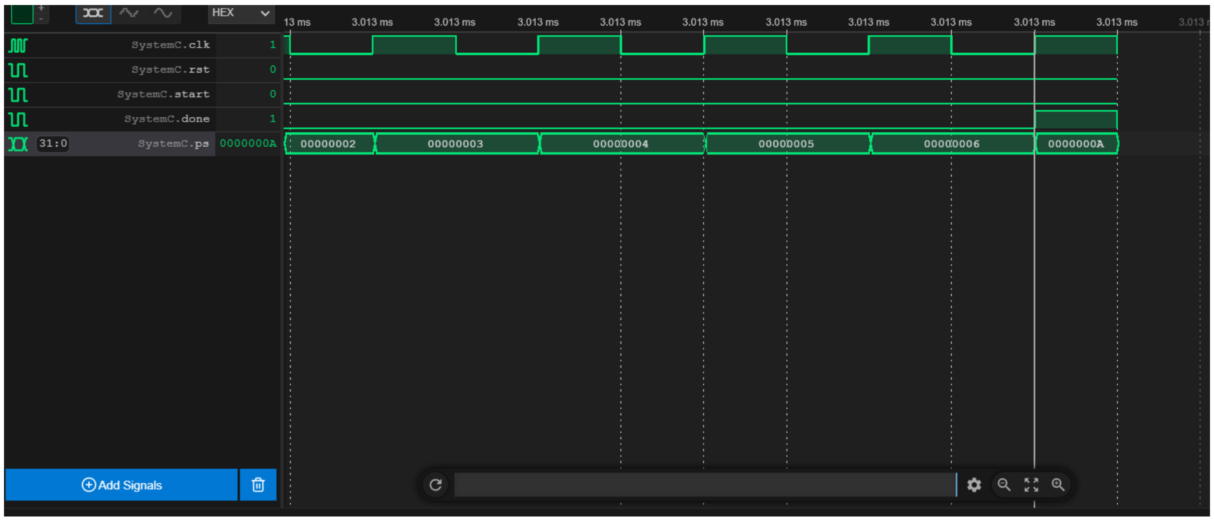
```
1111010110110100
1001101111110110
0110101011010100
1001101011110111
0101100011110110
1110010100110011
1001000000110010
1000101101001010
0111010100101010
0100000000101011
1100111001001111
1100001100101010
1010110101101001
1100111110010011
0100000110101010
0000100110101101
0000010010100111
1100010011011101
1101010111101001
0011110001101000
1111110010010110
1101110111000100
0111000100000111
0010001101110100
0101000001110010
```

```
0000000001010000
0000000010101000
0000000100101110
0000000100111001
0000000100111101
0000000110010100
0000001000101010
0000001101000010
0000010000000100
0000010010100111
0000010010101110
0000011011001010
0000011110011100
0000100000101010
0000100110010011
0000100110101101
0000100111001111
0000101000111001
0000101000111011
0000101101111110
0000101110010101
0000110000001000
0000110011010110
0000110100101101
0000110110000101
```

Fig. 6,7

SystemC Impementation:

The whole process is almost the same in systemC with some exception. In systemC there aren't any evl function for modules so we don't worry about the concurrency of our design components. I implemented each file with its corresponding file and structure in systemC even using the same file names. Just for components.h I also created a cpp file just for some beauty reason and make my code more readable. The result of my systemC simulation using VCD files is shown below:



As is obvious the done signal is issued when the memory sorting is completely done.