Jonathan W. Valvano

This is the closed book section. You must put your answers in the boxes on this answer page. You have 90 min, so please allocate your time accordingly. ***Please read the entire exam before starting.***

(4) Problem 1. Choose A-F	B) Temp allocation, private scope
(4) Problem 2. Choose A-H	D) Input capture on the rising edge of the digital signal
(4) Question 3. Choose A-D	**D)** None of the above
(4) Question 4. Choose A-D	**A)** $V_{OH} \geq V_{IH}$, $\lvert I_{OH}\rvert \geq \lvert I_{IH}\rvert$
(4) Question 5. Choose A-F	**E**
(4) Question 6. ADC bits	**20,000 alternatives requires 15 bits**
(4) Question 7. Yes or no, if no give example	**yes**
(4) Question 8. Yes or no, if no give example	**No,** **(10*10)/20 = 5** **10*(10/20) = 0**
(4) Question 9. Choose A-F	E) This will cause a run-time crash because the software does not clear **`RDRF`**

(2) Question 10. Choose A-Z,AA-JJ	D) bandwidth
(2) Question 11. Choose A-Z,AA-JJ	P) latency
(2) Question 12. Choose A-Z,AA-JJ	II) vectored interrupt
(2) Question 13. Choose A-Z,AA-JJ	W) private
(2) Question 14. Choose A-Z,AA-JJ	Q) minimally intrusive
(2) Question 15. Choose A-Z,AA-JJ	H) buffered I/O
(2) Question 16. Choose A-Z,AA-JJ	B) asynchronous serial
(2) Question 17. Choose A-Z,AA-JJ	O) half-duplex
(2) Question 18. Choose A-Z,AA-JJ	A) accuracy
(2) Question 19. Choose A-Z,AA-JJ	DD) resolution

(4) Question 20. There are 10 points to the IEEE Code of Ethics. What is the basic premise of the first point of this code? Give one specific example of how this might apply to embedded systems.

Take responsibility. When a bug or design flaw is found, make effort to fix it and to let the consumers know of the consequences of the error.

(5) Question 22. The actual throughput is 12 bits/2ms, which equals 6000 bits/sec.

(10) Question 23. Interface the following 16K ROM to a 6811 running at 2 MHz.
Synchronized negative logic, activate PROM when R/W=1, A15=0, and A14=1
(5) Part a) Design the interface between the ROM to the 6811.
Step 1. Design the address decoder
$4000-$7FFF is 01xx,xxxx,xxxx,xxxx
For fully decoded, specify all 0s and 1s. Select = not(A15)•A14 in positive logic

Step 2. Create a status table. The status table always starts like this

Select	R/W	Control Signals	Explanation
0	0		write cycle to another device
0	1		read cycle from another device
1	0		write cycle to our device
1	1		read cycle from our device

In this situation there is one negative-logic control signal called CE/

Select	R/W	CE/	Explanation
0	0		write cycle to another device
0	1		read cycle from another device
1	0		write cycle to our device
1	1		read cycle from our device

Basically in the status table you make the memory do the necessary functions. In this case we want to turn off the device (CE/ = 1) if the cycle is accesses another device. We will also turn it off if a write cycle occurs.

Select	R/W	CE/	Explanation
0	0	**1**	write cycle to another device
0	1	**1**	read cycle from another device
1	0	**1**	write cycle to our device
1	1		read cycle from our device

Finally, we will activate it if there is a read cycle to our device

Select	R/W	CE/	Explanation
0	0	1	write cycle to another device
0	1	1	read cycle from another device
1	0	1	write cycle to our device
1	1	**0**	read cycle from our device

Step 3. During the timing analysis we have to do two things. First, guarantee RDA overlaps RDR. And, second we have to make sure the memory doesn't drive the data bus during the first half of the cycle (because the 6811 is driving the low address during the first half of the cycle. To prevent the memory data from colliding we will only drive data out of the memory when E=1. This is called synchronizing the read operation to E. To create a combined table, we expand the status table, adding the E as an input. The data from the status table in entered in the positions where E=1.

E	Select	R/W	CE/	Explanation
0	0	0		
1	**0**	**0**	**1**	**write cycle to another device**
0	0	1		
1	**0**	**1**	**1**	**read cycle from another device**
0	0	0		
1	**1**	**0**	**1**	**write cycle to our device**
0	1	1		
1	**1**	**1**	**0**	**read cycle from our device**

To make the control signal high throughout the cycles when we wish to disable the memory, we place additional ones

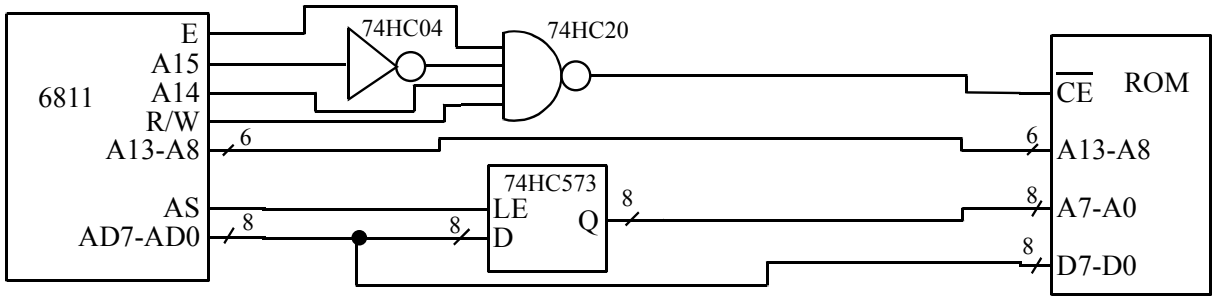
E	Select	R/W	CE/	Explanation
0	0	0	**1**	
1	0	0	1	write cycle to another device
0	0	1	**1**	
1	0	1	1	read cycle from another device
0	0	0	**1**	
1	1	0	1	write cycle to our device
0	1	1		
1	1	1	0	read cycle from our device

To make the control signal synchronized negative logic, we place a 1,0 in those entries for the cycle we wish to activate. See the book Figure 9.28 and Table 9.9 for the choices that can be entered into the combined table.

E	Select	R/W	CE/	Explanation
0	0	0	1	
1	0	0	1	write cycle to another device
0	0	1	1	
1	0	1	1	read cycle from another device
0	0	0	1	
1	1	0	1	write cycle to our device
0	1	1	**1**	
1	1	1	**0**	read cycle from our device

Step 4. Develop the logic equation for the control signal and build it with real gates

CE/ = not(A15)•A14•R/W•E



(5) Part b) To show that the timing requirements are satisfied, we write equations for RDA and RDR. The gate delay is [5ns min, 10ns max].

RDA = (fall of CE/+[60,80], rise of CE/+10)

Because CE/ is synchronized to the E clock, the fall of CE/ will be 250+gate delay = [255,260]. Similarly, the rise of CE/ will be 500+gate delay = [505,510].

RDA = ([255,260]+[60,80], [505,510]+10) = (340,515) *choose shortest RDA interval*

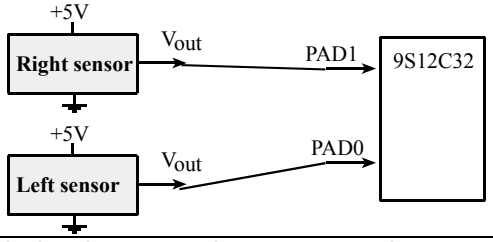
RDR comes from the 6811. At 2 MHz it is

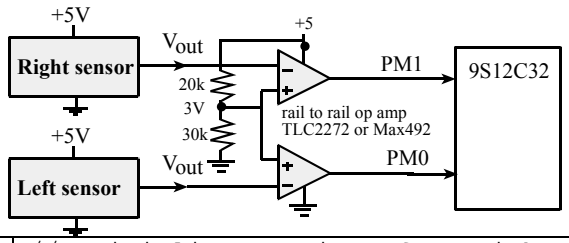
RDR = (450,510)

Notice that RDA overlaps RDR

(25) Problem 24. The goal is to design the robot so it follows the black line on the track.

(10) Part a) You could use a threshold detector (more expensive) or the ADC to interface the sensor.



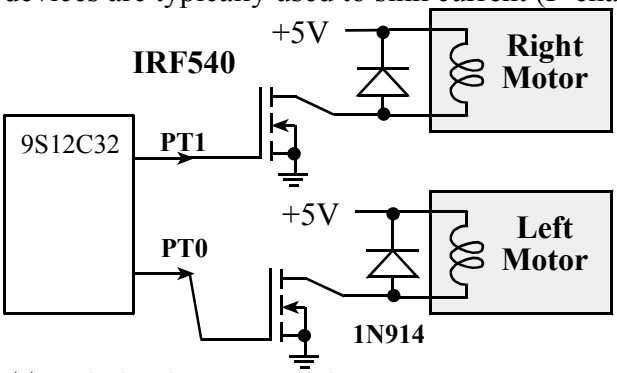


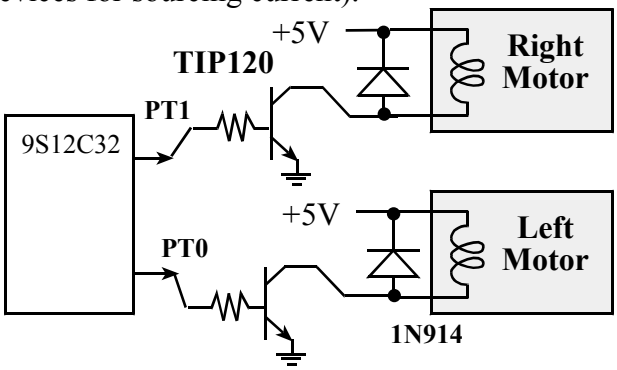

```
// Initialize ADC interface with sensor
void Sensor_Init(void){
  ATDCTL2 = 0x80; // enable ADC
  ATDCTL3 = 0x10; // Sequence length = 2
  ATDCTL4 = 0x05; // 10-bit resolution
}
// 0 if both off, 1 if left on track
// 2 if right on, 3 if both on track
unsigned short Sensor_Read(void){
unsigned short result=0;
  ATDCTL5 = 0x90; // sequence PAD0, PAD1
  while((ATDSTAT1&0x02)==0){};  // CCF1
  if( ATDDR0 < 614 ) result = 1;
  if( ATDDR1 < 614 ) result |= 2;
  return result;
}
```

```
// Initialize PTM interface with sensor
void Sensor_Init(void){
  DDRM &= ~0x03; // PM1 PM0 inputs
}


// 0 if both off, 1 if left on track
// 2 if right on, 3 if both on track
unsigned short Sensor_Read(void){
  return PTM&0x03;
}
```

(10) Part b) Because of the 2A, I will use a MOSFET or a Darlington to switch the current. N-channel devices are typically used to sink current (P-channel devices for sourcing current).

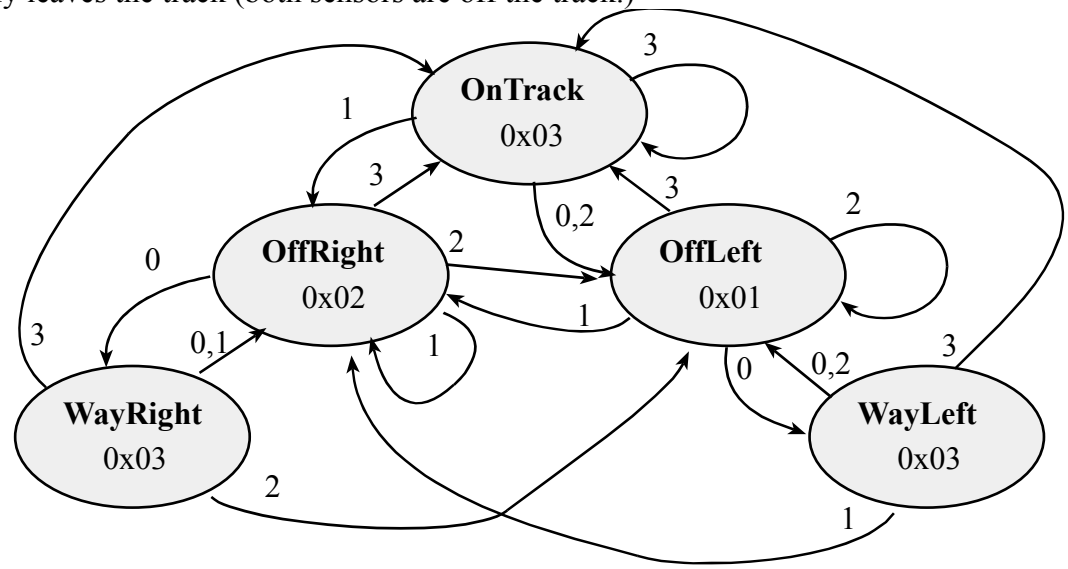





```
// Initialize Motor interface
void Motor_Init(void){
  DDRT |= 0x03; // PT1 PT0 outputs
}
```

```
// 0 stop both,
// 1 active left (turns right)
// 2 active right (turns left),
// 3 means activate both motors (causing the robot to move forward).
void Motor_Out(unsigned char controlValue){
  PTT = (PTT&0xFC)+controlValue;
}
```

(5) Part c) Output=1 to turn right, output=2 to turn left. Input=1 if right side is off track, input=2 if left side is off track. If the car is on the track go straight. If the car is off to the right, turn left. If the car is off to the left, turn right. Run the algorithm about 10 times faster than the response time of the motors. A finite state machine is needed to solve this problem in order to handle the case where the car completely leaves the track (both sensors are off the track.)



Simulation of this algorithm.

The robot car is moving clockwise around the green track. The red dots show positions the places the robot car has traveled.

```
OnTrack
      fcb  %11    ;move both wheels
      fdb  OffLeft,OffRight,OffLeft,OnTrack
OffLeft
      fcb  %01    ;move just the right wheel
      fdb  WayLeft,OffRight,OffLeft,OnTrack
WayLeft
      fcb  %11    ;move both wheels
      fdb  OffLeft,OffRight,OffLeft,OnTrack
OffRight
      fcb  %10    ;move just the left wheel
      fdb  WayRight,OffRight,OffLeft,OnTrack
WayRight
      fcb  %11    ;move both wheels
      fdb  OffRight,OffRight,OffLeft,OnTrack
```