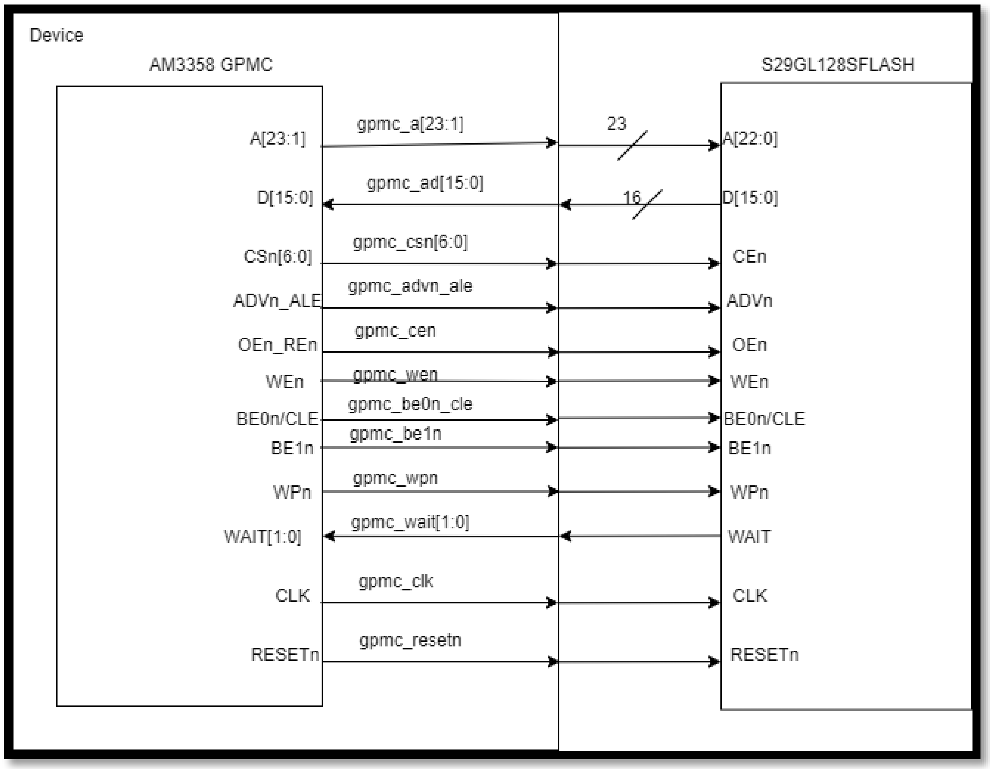
Homework 4

Swati Kadivar

1. Memory Addressing

a. Design the asynchronous non-multiplexed bus interface between a Spansion S29GL128SFLASH memory and TI AM3358 MPU, by consulting datasheet and reference manuals for both.

b. Draw the connections between the devices, showing connections to the FLASH chip select CE#, write enable WE#, output enable OE#, write protect WP#, RESET#, address bus A0-A22 and data bus DQ0-DQ15 from the processor. You may use a CAD tool, or draw the schematic for the 2 chips by hand.

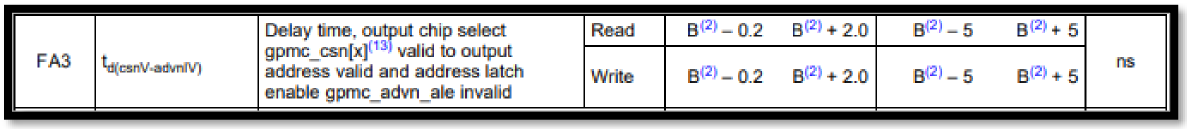


c. Draw the timing diagram for a read cycle, showing the FLASH chip select CE#, write enable WE#, output enable OE#, address bus A0-A22 and data bus DQ0-DQ15signals. Assume the processor runs at 100 MHz for the GPMC FCLK clock.

Calculating all the parameters for given case:

The AccessTime is roundmax (FLASH tACC + Setup)/GPMC_FCLK = (100+10)/10 = 11 clock cycles, 110ns.

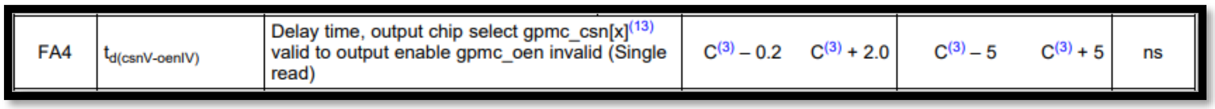
FA1 (chip select low) = A = (CSRdOffTime – CSOnTime) x (TimeParaGranularity +1) x GPMC_FCLK = (12-0) x (0-1) x 10ns = 120ns, as CSRdOffTime is the AccessTime + 1 = 11 and CSOnTime is 0 from Table 7-46 in TRM.



FA3	$t_{d(csnV-advnIV)}$	Delay time, output chip select gpmc_csn[x][13] valid to output address valid and address latch enable gpmc_advn_ale invalid	Read	B[2] – 0.2	B[2] + 2.0	B[2] – 5	B[2] + 5	ns
			Write	B[2] – 0.2	B[2] + 2.0	B[2] – 5	B[2] + 5	

FA3 = ((ADVRdOffTime – CSOnTime) x (1) + 0.5 x (ADVExtraDelay – CSExtraDelay)) x GPMC_FCLK + 2.0

= ((1-0) x (1) +0.5 x (0-0)) x 10 ns + 2.0 = **12ns**



FA4	$t_{d(csnV-oenIV)}$	Delay time, output chip select gpmc_csn[x][13] valid to output enable gpmc_oen invalid (Single read)	C[3] – 0.2	C[3] + 2.0	C[3] – 5	C[3] + 5	ns

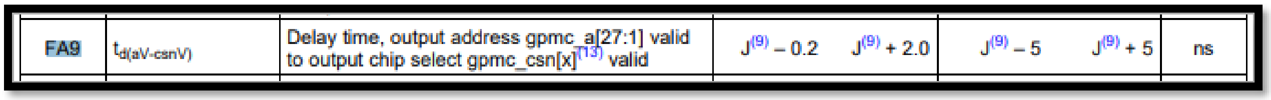
FA4 = ((OEOffTime – CSOnTime) x (TimeParaGranularity + 1) +0.5 x (OEExtraDelay – CSExtraDelay)) x GPMC_FCLK + 2.0

= ((12-0) x (0+1) +0.5 x (0 – 0)) x GPMC_FCLK + 2.0 = **122ns**

FA5 parameter illustrates amount of time required to internally sample input data. It is expressed in number of GPMC functional clock cycles. From start of read cycle and after FA5 functional clock cycles, input data will be internally sampled by active functional clock edge. FA5 value must be stored inside AccessTime register bits field.

FA5 = AccessTime x (TimeParaGranularity + 1) x GPMC_FCLK

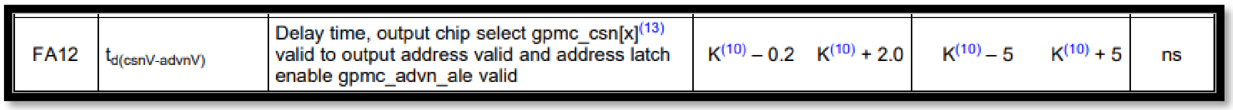
= 11 x (0 + 1) x 10 = **110 ns**



FA9	$t_{d(aV-csnV)}$	Delay time, output address gpmc_a[27:1] valid to output chip select gpmc_csn[x][13] valid	J[9] – 0.2	J[9] + 2.0	J[9] – 5	J[9] + 5	ns

FA9 = (CSOnTime x (TimeParaGranularity + 1) + 0.5 x CSExtraDelay) x GPMC_FCLK + 2.0

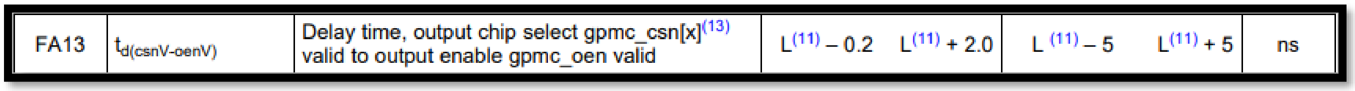
= (0 x (0+1) + 0.5 x 0) x GPMC_FCLK + 2.0 = **2ns**



FA12	$t_{d(csnV-advnV)}$	Delay time, output chip select gpmc_csn[x][13] valid to output address valid and address latch enable gpmc_advn_ale valid	K[10] – 0.2	K[10] + 2.0	K[10] – 5	K[10] + 5	ns

FA12 = ((ADVOnTime – CSOnTime) x (TimeParaGranularity + 1) + 0.5 x (ADVExtraDelay – CSExtraDelay)) x GPMC_FCLK + 2.0

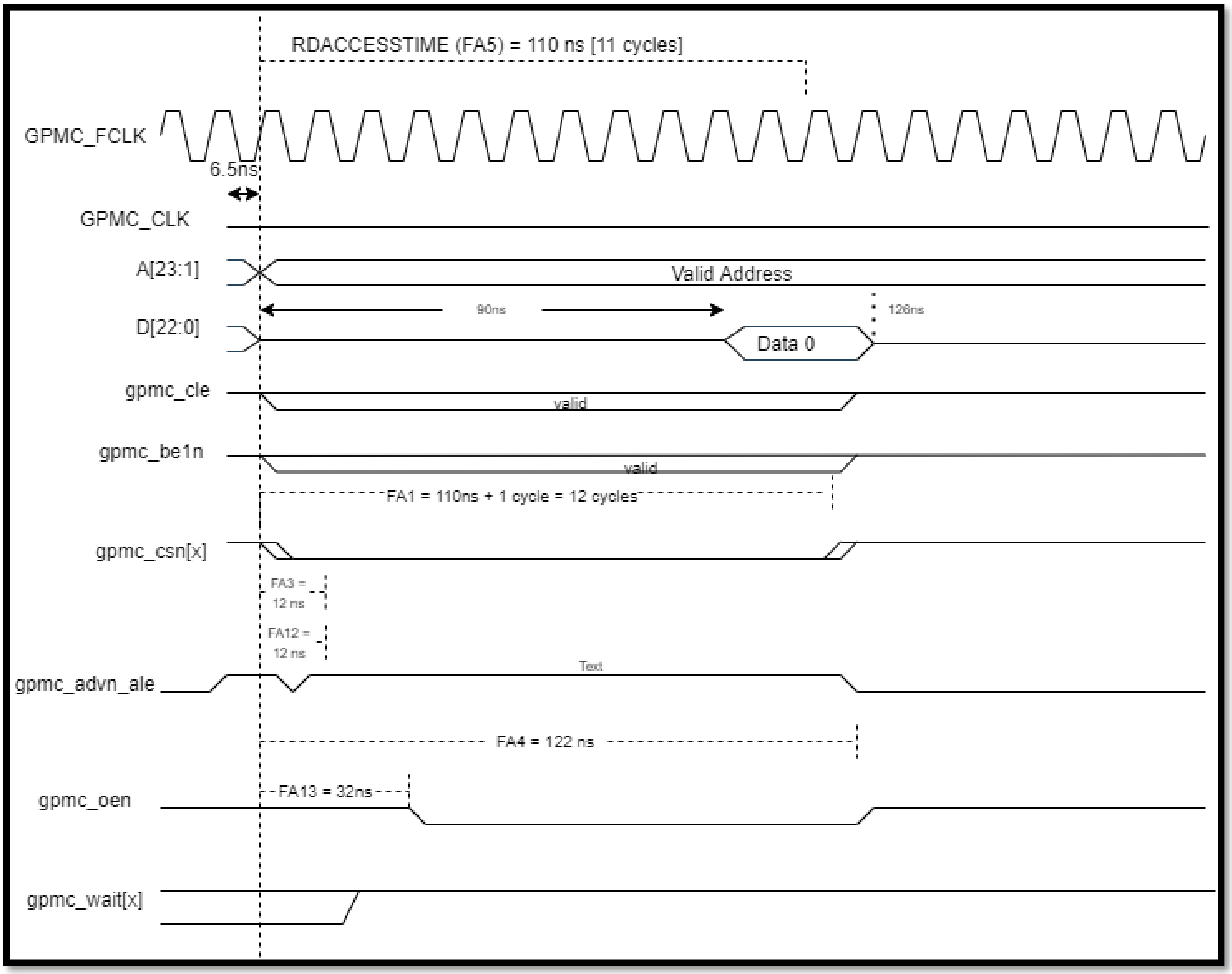
= ((1-0) x (0+1) +0.5 x (0-0)) x GPMC_FCLK + 2.0 = **12ns**



FA13	$t_{d(csnV-oenV)}$	Delay time, output chip select gpmc_csn[x][13] valid to output enable gpmc_oen valid	L[11] – 0.2	L[11] + 2.0	L[11] – 5	L[11] + 5	ns

F13 = ((OEOnTime – CSOnTime) x (TimeParaGranularity + 1) + 0.5 x (OEExtraDelay – CSExtraDelay)) x GPMC_FCLK + 2.0

= ((3-0) x (0+1) + 0.6 x (0-0)) x GPMC_FCLK + 2.0 = **32ns**



d. From the timing diagram, determine the number of wait states required and write it here: __9__

Wait states = Access time – minimum access time = 11-2 = 9

e. List the configuration registers that must be determined and programmed in the AM3358 for this interface to work.

Minimum GPMC_CONFIG1_i, GPMC_CONFIG2_i, GPMC_CONFIG3_i, GPMC_CONFIG4_i, GPMC_CONFIG5_i, GPMC_CONFIG6_i and GPMC_CONFIG7_i.

2. Baud Rates

a. What is the difference between a bit rate and a baud rate? Give an example in which they are not the same.

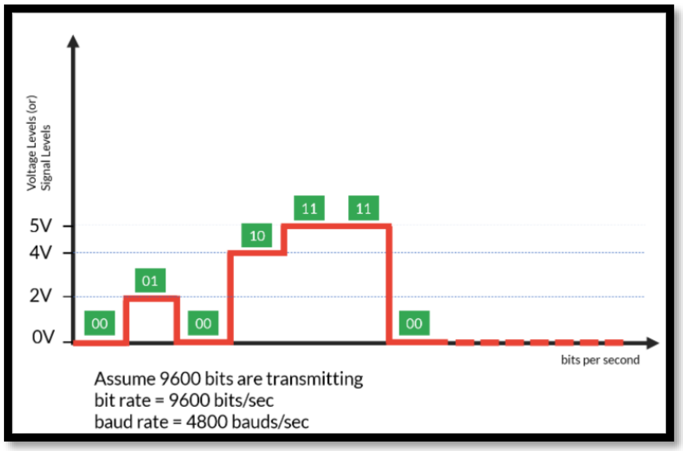
- Both Bit rate and Baud rate are generally used in data communication,
- Bit rate is the transmission of number of bits per second. On the other hand, Baud rate is defined as the number of signal units per second. The formula which relates both bit rate and baud rate is given below:

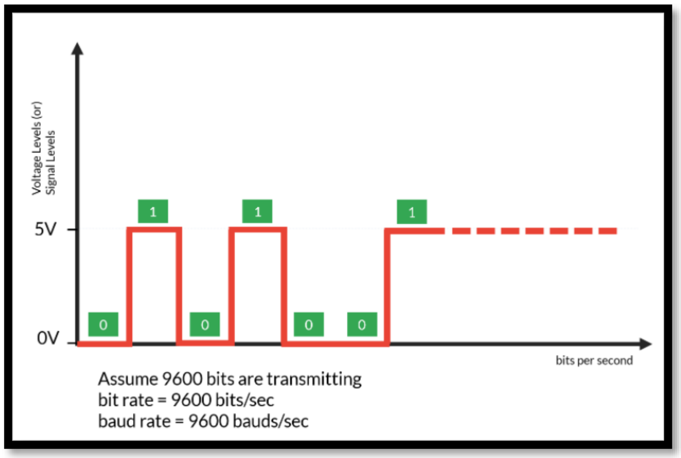
Bit rate = Baud rate x the number of bits per baud

Bit Rate	Baud Rate
Bit rate is defined as the transmission of number of bits per second.(bits/s)	Baud rate is defined as the number of signal units per second.(baud/s)
Bit rate is also defined as per second travel number of bits.	Baud rate is also defined as per second number of changes in signal.
Bit rate emphasized on computer efficiency.	While baud rate emphasized on data transmission.
The formula of Bit Rate is: = baud rate x the number of bits per baud	The formula of Baud Rate is: = bit rate / the number of bits per baud
Bit rate is not used to decide the requirement of bandwidth for transmission of signal.	While baud rate is used to decide the requirement of bandwidth for transmission of signal.

- When we talk about binary signaling (high – 1 or 5V and low – 0 or 0V), the bit rate and baud rate mean the same thing. Number of bits transferred is equal to number of state changes.
- As most of the time binary signaling is being used, bit rate and baud rate are mistaken to be the same thing.
- In multilevel signaling, there are more than two voltage or signal levels. This is because a greater number of bits are required to represent the different levels of voltage. The number of bits required to represent m signals or voltage is N = log2(m). So in that case, bit rate is going to be higher than baud rate.

Examples:



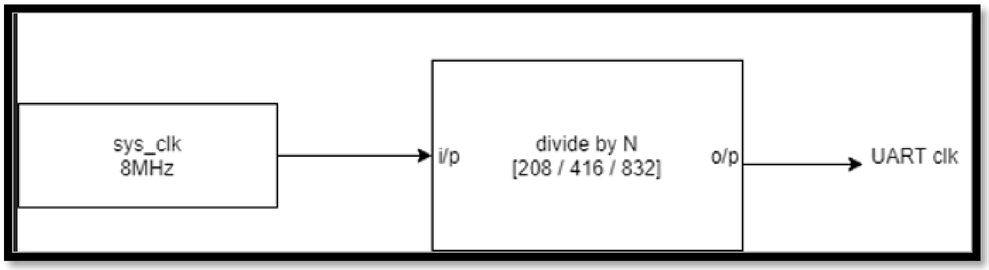


Ref:

https://bytesofgigabytes.com/embedded/bit-rate-and-baud-rate/

https://www.geeksforgeeks.org/difference-btween-bit-rate-and-baud-rate/

b. If you are operating with a system clock of 8.000 MHz +/- 0.01 %, at what baud rates can a divide by n baud rate generator provide reliable communication for a UART? Work your way down from a baud rate of 921600. Determine n, the divider for 3 common baud rates using this clock.



- As shown in above diagram, if we are operating with a system clock of 8MHz +/-0.01%, I would choose Divide by N factor for baud rate generator as 208. To achieve other baud rates, we can use additional divide by 2's and can achieve three standard baud rates of 38400, 19200, 9600.
- To achieve proper baud rate, we can use the system clock as 7,987,200 Hz.

Difference % = (8000000-7987200)/8000000 = 0.0016 which satisfies our requirement of diff % of 8MHz +/- 0.01%.

1. **N = 208**

Baud rate = system clk frequency / N = 7987200/208 = 38400

2. **N = 416**

Baud rate = system clk frequency / N = 7987200/416 = 19200

3. N = 832

Baud rate = system clk frequency / N = 7987200/832 = 9600

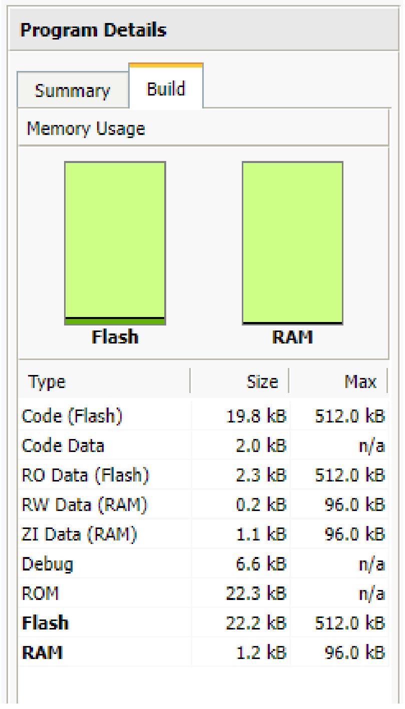
N	System clock	Baud Rate
208	7987200	38400
416	7987200	19200
832	7987200	9600

3. Saturating Arithmetic

The ARM Cortex-M4 has saturating arithmetic assembly language functions, which should give it a performance advantage when doing math compared to the M0+. This type of math is useful in general, but also particularly when implementing block floating point. In saturating arithmetic, any overflow is set to the maximum allowable value and any underflow is set to the lowest value. The question is, does the compiler make use of these special assembly language instructions?

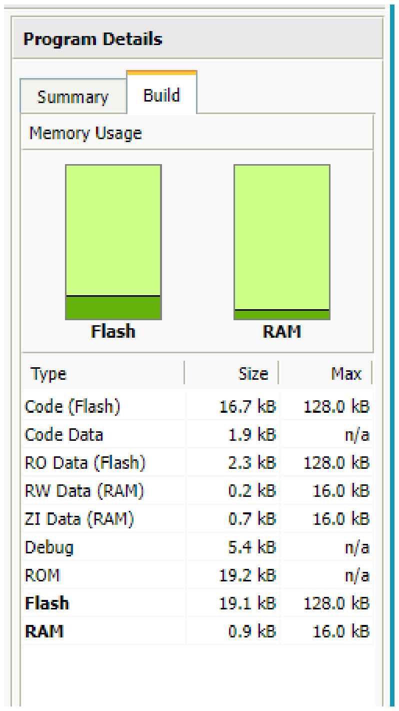
a. Implement in C a function(s) to do saturated unsigned addition for 8, 16, and 32 bit numbers. Compile this for both the M4 (ST) and M0+ (Freescale). Compare the code size of the compilations. Which is smaller?

NUCLEO-F401RE - ADD



Type	Size	Max
Code (Flash)	19.8 kB	512.0 kB
Code Data	2.0 kB	n/a
RO Data (Flash)	2.3 kB	512.0 kB
RW Data (RAM)	0.2 kB	96.0 kB
ZI Data (RAM)	1.1 kB	96.0 kB
Debug	6.6 kB	n/a
ROM	22.3 kB	n/a
Flash	22.2 kB	512.0 kB
RAM	1.2 kB	96.0 kB

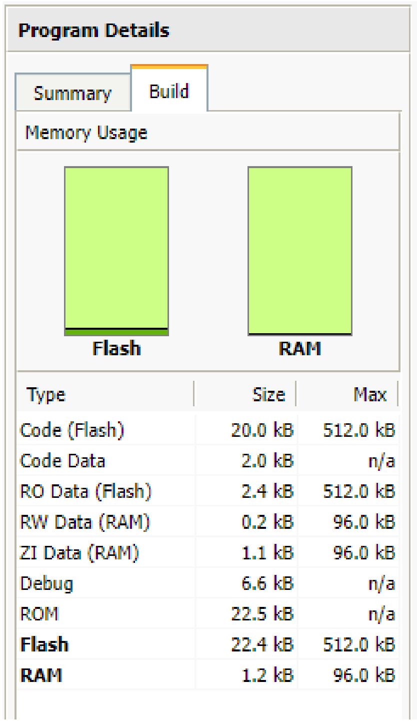
FRDM-KL25Z - ADD



Type	Size	Max
Code (Flash)	16.7 kB	128.0 kB
Code Data	1.9 kB	n/a
RO Data (Flash)	2.3 kB	128.0 kB
RW Data (RAM)	0.2 kB	16.0 kB
ZI Data (RAM)	0.7 kB	16.0 kB
Debug	5.4 kB	n/a
ROM	19.2 kB	n/a
Flash	19.1 kB	128.0 kB
RAM	0.9 kB	16.0 kB

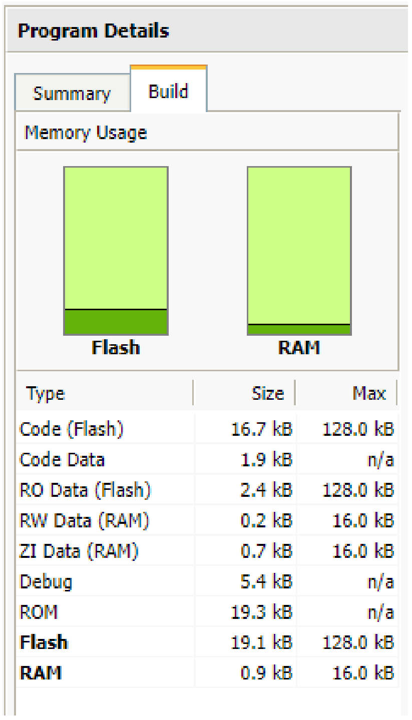
b. Repeat (a) for saturated unsigned multiplication.

NUCLEO-F401RE - MUL



Type	Size	Max
Code (Flash)	20.0 kB	512.0 kB
Code Data	2.0 kB	n/a
RO Data (Flash)	2.4 kB	512.0 kB
RW Data (RAM)	0.2 kB	96.0 kB
ZI Data (RAM)	1.1 kB	96.0 kB
Debug	6.6 kB	n/a
ROM	22.5 kB	n/a
Flash	22.4 kB	512.0 kB
RAM	1.2 kB	96.0 kB

FRDM-KL25Z - MUL

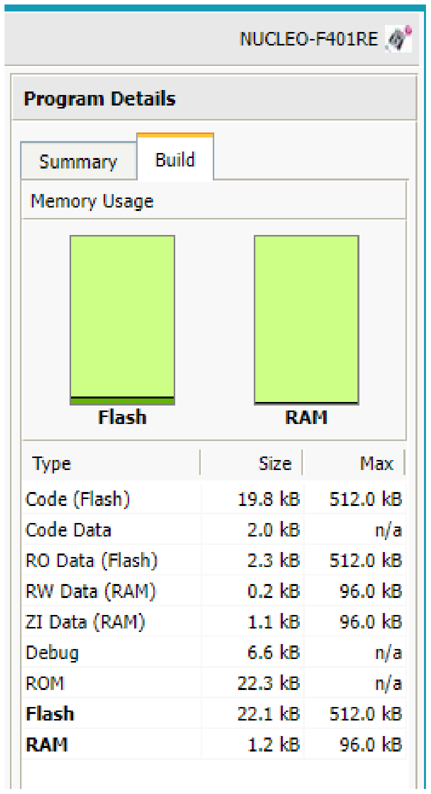


Type	Size	Max
Code (Flash)	16.7 kB	128.0 kB
Code Data	1.9 kB	n/a
RO Data (Flash)	2.4 kB	128.0 kB
RW Data (RAM)	0.2 kB	16.0 kB
ZI Data (RAM)	0.7 kB	16.0 kB
Debug	5.4 kB	n/a
ROM	19.3 kB	n/a
Flash	19.1 kB	128.0 kB
RAM	0.9 kB	16.0 kB

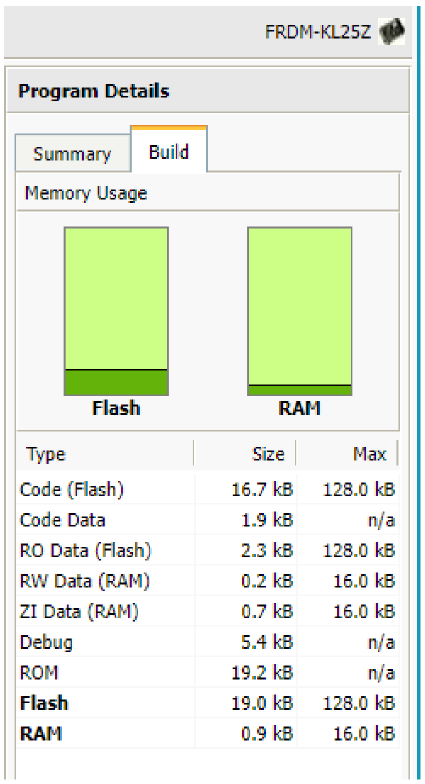
c. Repeat (a) for saturated signed subtraction.

NUCLEO-F401RE - SUB

FRDM-KL25Z - SUB



Type	Size	Max
Code (Flash)	19.8 kB	512.0 kB
Code Data	2.0 kB	n/a
RO Data (Flash)	2.3 kB	512.0 kB
RW Data (RAM)	0.2 kB	96.0 kB
ZI Data (RAM)	1.1 kB	96.0 kB
Debug	6.6 kB	n/a
ROM	22.3 kB	n/a
Flash	22.1 kB	512.0 kB
RAM	1.2 kB	96.0 kB



Type	Size	Max
Code (Flash)	16.7 kB	128.0 kB
Code Data	1.9 kB	n/a
RO Data (Flash)	2.3 kB	128.0 kB
RW Data (RAM)	0.2 kB	16.0 kB
ZI Data (RAM)	0.7 kB	16.0 kB
Debug	5.4 kB	n/a
ROM	19.2 kB	n/a
Flash	19.0 kB	128.0 kB
RAM	0.9 kB	16.0 kB

Please find the code for above compilation below in Appendix I.

I am observing that code on FRDM-KL25Z is occupying less size after compilation than NUCLEO-F401RE.

4. Saturating Arithmetic

Implement C-callable assembly language functions to do saturated unsigned addition for 8, 16, and 32 bit numbers. Make use of saturated instructions. Compile this for the M4 (ST), and note the code size of the compilation. Repeat for saturated unsigned multiplication and signed subtraction.

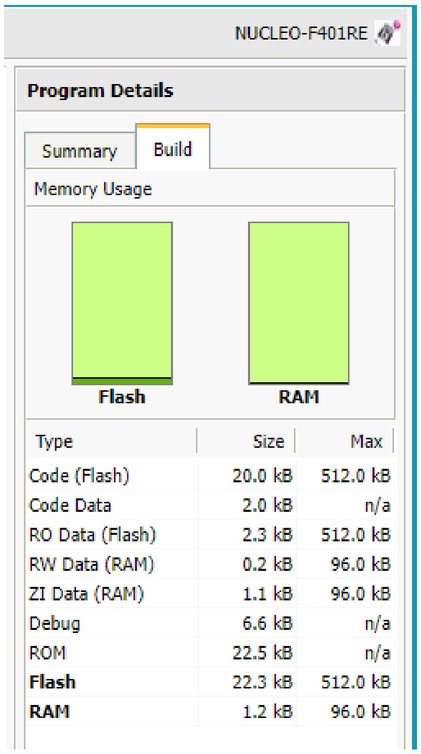
Unsigned Addition:

```
__asm uint32_t sat_add_U8(uint32_t a, uint32_t b)
{
   UQADD8 r3, r0, r1
   MOVS r0, r3
   bx lr
}
```

```
__asm uint32_t sat_add_U16(uint32_t a, uint32_t b)
{
   UQADD16 r3, r0, r1
   MOVS r0, r3
   bx lr
}

__asm uint32_t sataddU32(uint32_t a, uint32_t b)
{
   QADD r3, r0, r1
   MOVS r0, r3
   bx lr
}

uint32_t sat_add_U32(uint32_t a, uint32_t b)
{
   uint32_t result = sataddU32(a, b);
   if(result > 0xFFFFFFFF)
      result = 0xFFFFFFFF;
   if(a>0 && b > 0 && result < a && result < b)
      result = 0xFFFFFFFF;
   return result;
}
```



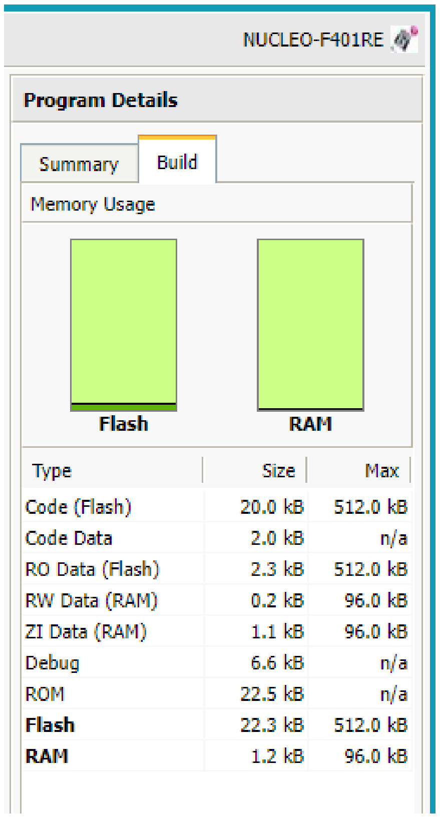
Type	Size	Max
Code (Flash)	20.0 kB	512.0 kB
Code Data	2.0 kB	n/a
RO Data (Flash)	2.3 kB	512.0 kB
RW Data (RAM)	0.2 kB	96.0 kB
ZI Data (RAM)	1.1 kB	96.0 kB
Debug	6.6 kB	n/a
ROM	22.5 kB	n/a
Flash	22.3 kB	512.0 kB
RAM	1.2 kB	96.0 kB

Unsigned Multiplication:

```
__asm uint32_t sat_mul_U8(uint32_t a, uint32_t b) {
   MULS r0, r1, r0
   CMN r0, #255
   BGT SAT
   bx lr
SAT MOVS r0, #255
   bx lr
}

__asm uint32_t sat_mul_U16(uint32_t a, uint32_t b, uint32_t c) {
   MULS r0, r1, r0
   CMP r0, r3
   BGT SAT1
   bx lr
SAT1 MOVS r0, r3
   bx lr
}

__asm uint32_t sat_mul_U32(uint32_t a, uint32_t b) {
   UMULL r0, r1, r0, r1
   bx lr
}
```



NUCLEO-F401RE

Program Details

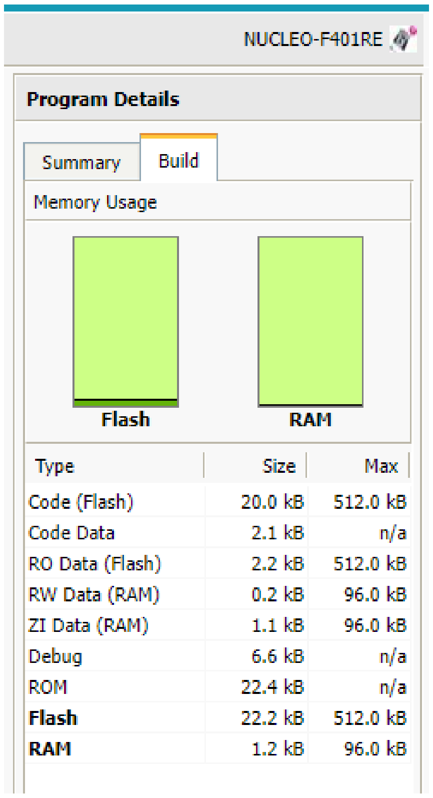
Type	Size	Max
Code (Flash)	20.0 kB	512.0 kB
Code Data	2.0 kB	n/a
RO Data (Flash)	2.3 kB	512.0 kB
RW Data (RAM)	0.2 kB	96.0 kB
ZI Data (RAM)	1.1 kB	96.0 kB
Debug	6.6 kB	n/a
ROM	22.5 kB	n/a
Flash	22.3 kB	512.0 kB
RAM	1.2 kB	96.0 kB

Signed Subtraction:

```
__asm uint32_t sat_sub_8(uint32_t a, uint32_t b)
{
   QSUB8 r3, r0, r1
   MOV r0, r3
   bx lr
}

__asm uint32_t sat_sub_16(uint32_t a, uint32_t b)
{
   QSUB16 r3, r0, r1
   MOV r0, r3
   bx lr
}

__asm uint32_t sat_sub_32bit(uint32_t a, uint32_t b)
{
```



NUCLEO-F401RE

Program Details

Type	Size	Max
Code (Flash)	20.0 kB	512.0 kB
Code Data	2.1 kB	n/a
RO Data (Flash)	2.2 kB	512.0 kB
RW Data (RAM)	0.2 kB	96.0 kB
ZI Data (RAM)	1.1 kB	96.0 kB
Debug	6.6 kB	n/a
ROM	22.4 kB	n/a
Flash	22.2 kB	512.0 kB
RAM	1.2 kB	96.0 kB

```
    QSUB r3, r0, r1
    MOV r0, r3
    bx lr
}
```

Calculations:

addition tests:
1. 0x80+0x80 = 255,
2. 0x00+0xFF = 255,
3. 0xFF+ 0xFF = 255,
4. 0xF1+0x0F = 255,
5. 0x8000+0x8000 = 65535,
6. 0x000F+0xFFF0 = 65535,
7. 0xFFFF+0xFFFF = 65535,
8. 0xFFF1+0x000F = 65535,
9. 0x80000000+0x80000000 = 4294967295,
10. 0xFFFFFFF1+0x0000000F = 4294967295

multiplication tests:
11. 0xFF x 0x02 = 255,
12. 0xFF x 0x01 = 255,
13. 0x0F x 0x0F = 225,
14. 0xFFFF*0x0002 = 65535,
15. 0xFFFF x 0x0001 = 65535,
16. 0x00FF x 0x00FF = 65025,
17. 0xFFFFFFFF x 0x00000002 = 4294967295,
18. 0xFFFFFFFF x 0x00000001 = 4294967295,
19. 0x0000FFFF x 0x0000FFFF = 4294836225,

subtraction tests:
20. 127-128 = -1,
21. 127-127 = 0,
22. -127 – (-127) = 0,
23. 127 – (-128) = 127,
24. 32767-32768 = -1,
25. -32767 – (-32767) = 0,
26. 32767 – (-32768) = 32767

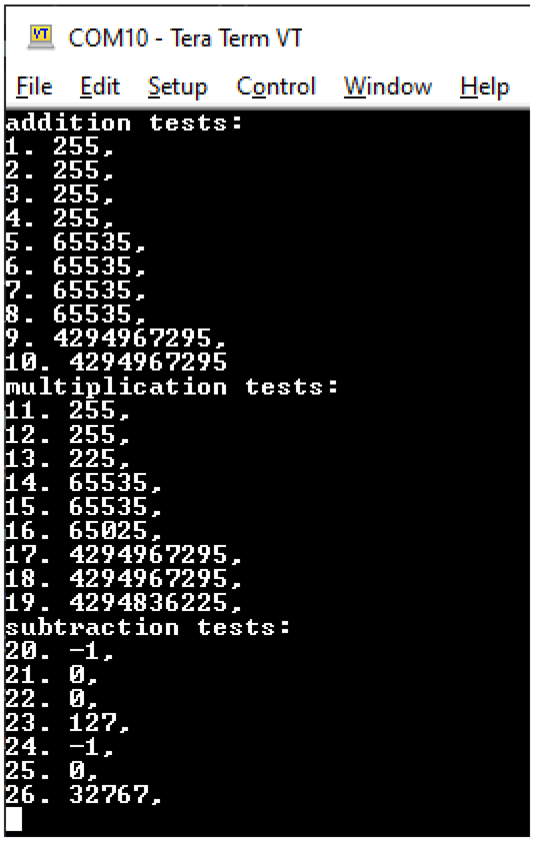
5. Saturating Arithmetic

Test your functions in the previous 2 problems with these numbers:

a. Addition: 0x80+0x80, 0x00+0xFF, 0xFF+ 0xFF, 0xF1+0x0F, 0x8000+0x8000, 0x000F+0xFFF0, 0xFFFF+0xFFFF, 0xFFF1+0x000F, 0x80000000+0x80000000, 0xFFFFFFF1+0x0000000F

b. Multiplication: 0xFF x 0x02, 0xFF x 0x01, 0x0F x 0x0F; 0xFFFF*0x0002, 0xFFFF x 0x0001, 0x00FF x 0x00FF, 0xFFFFFFFF x 0x00000002, 0xFFFFFFFF x 0x00000001, 0x0000FFFF x 0x0000FFFF

c. Subtraction: 127-128, 127-127, -127 – (-127), 127 – (-128); 32767-32768, -32767 – (-32767), 32767 – (-32768)



```
addition tests:
1. 255,
2. 255,
3. 255,
4. 255,
5. 65535,
6. 65535,
7. 65535,
8. 65535,
9. 4294967295,
10. 4294967295
multiplication tests:
11. 255,
12. 255,
13. 225,
14. 65535,
15. 65535,
16. 65025,
17. 4294967295,
18. 4294967295,
19. 4294836225,
subtraction tests:
20. -1,
21. 0,
22. 0,
23. 127,
24. -1,
25. 0,
26. 32767,
```

Please find the code for above test cases in Appendix I.

Review 4.3 What are the two major categories of memory device that are utilized in embedded applications?

Understanding the characteristics of each and how to design with them can lead to more robust and effective design solutions. We classify memory into two general categories: ROM and RAM. We further subdivide RAM into static RAM (SRAM) and dynamic RAM (DRAM). These classifications lead to several different subcategories of both ROM and RAM.

Review 4.21 What kinds of information are typically identified in a memory map?

At a bare minimum, the memory map should identify the data and code space. the memory map lists the addresses in memory allocated to each portion of the application.

Review 4.23 What kinds of memory devices would one typically find in each of the categories in the memory hierarchy discussed in the chapter?

Memories in the hierarchy are related by hierarchical metrics like speed and storage capacity. At the top are the slowest, largest, and least expensive memories. These are known as **secondary memory** and are shown in the diagram by the block on the left. For example, disks. At the bottom are the **smallest, fastest memories called cache memory**; these are typically higher speed **SRAMs**. These devices also tend to be the most expensive. In the middle of the hierarchy is main or primary memory. These are either **lower speed SRAM devices or, more commonly, DRAM memories**. CPU registers are sometimes included in the ranking as higher speed memory than cache.

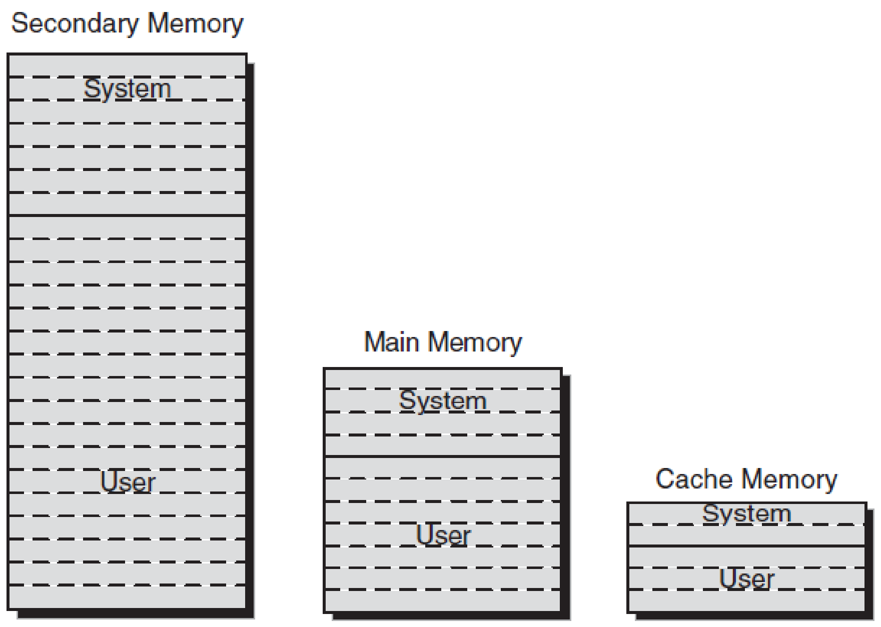
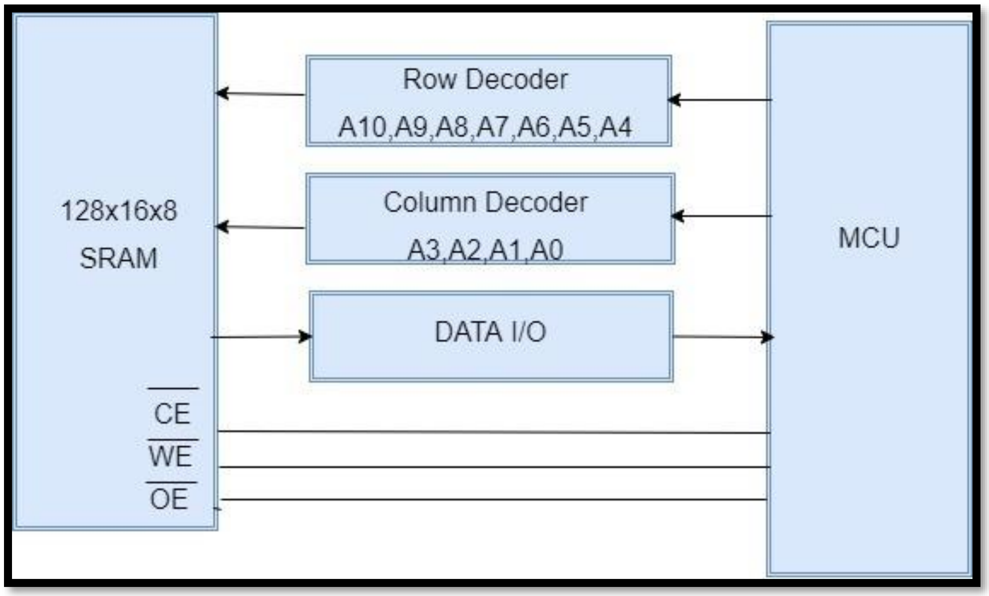


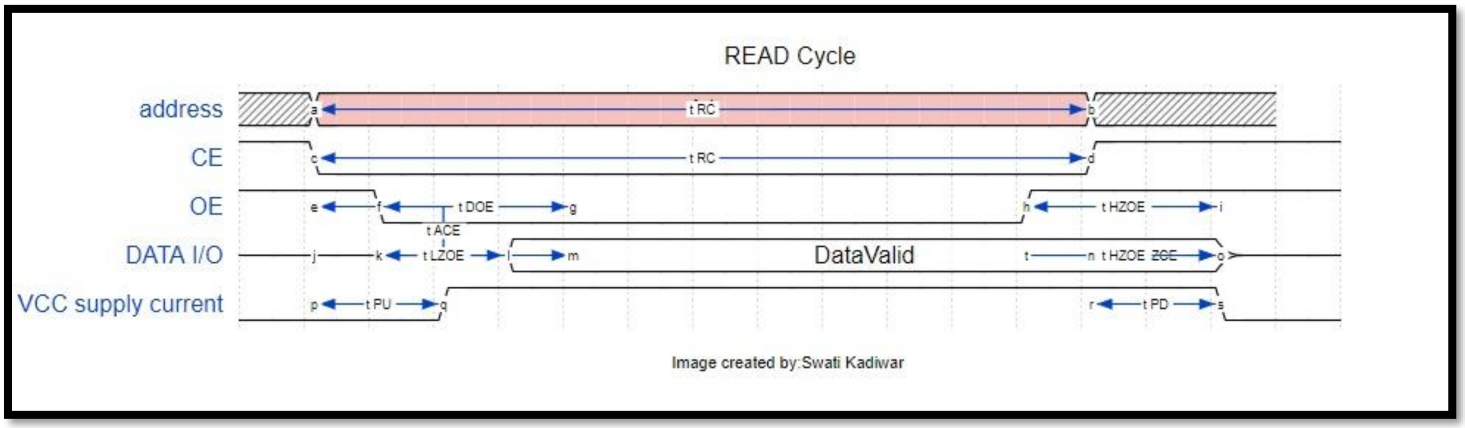
Figure 4.32 Typical Memory Hierarchy Utilizing a Variety of Memory Types

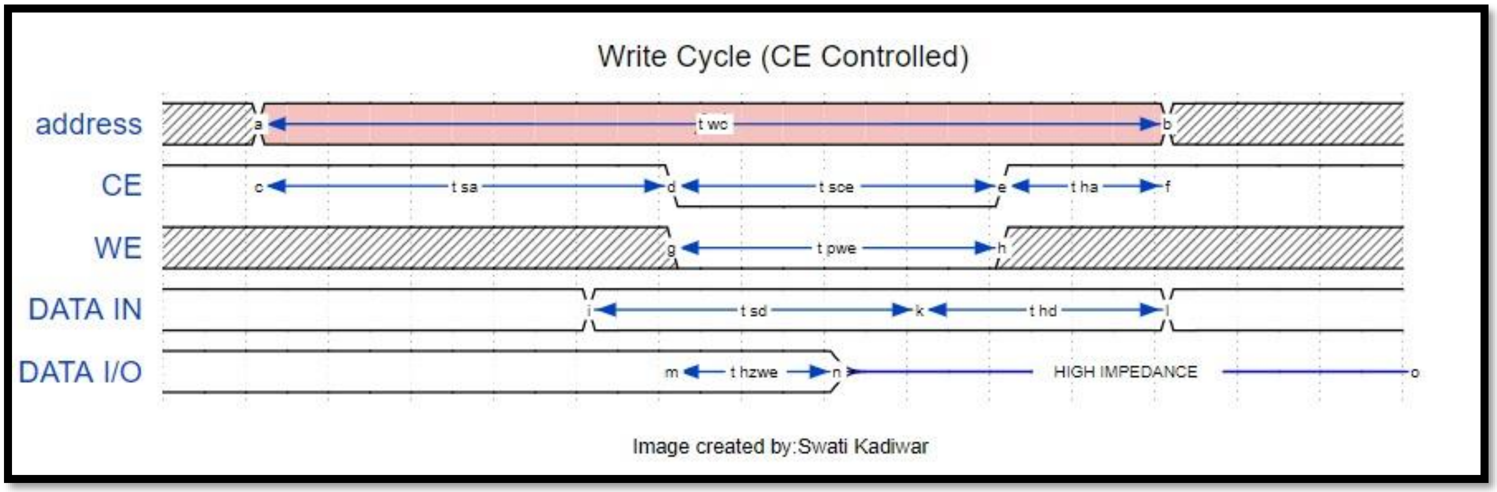
Problem 4.1 A memory system is needed in a new design to support a small amount of data storage outside of the processor. The design is to be based on the 16 K bit CY7C128A SRAM organized as 2 K x 8.

(a) Provide a high-level block diagram for such an interface.

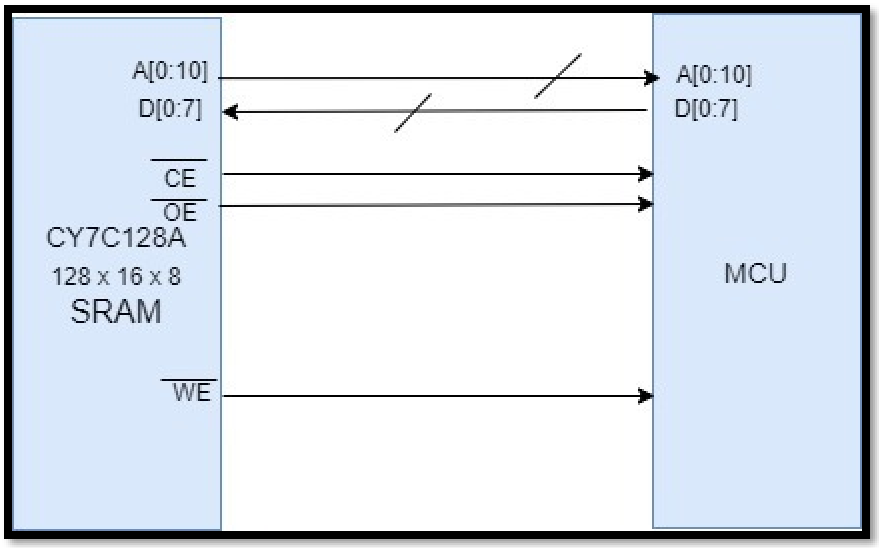


(b) Provide a high-level timing diagram for the interface to the SRAM from the microprocessor, assuming that separate address and data busses are available. Define any control signals that may be necessary.





(c) Design the interface based on the timing diagram from part (a).



(d) Analyze the memory performance for a write and a read operation of 1, 10, and 100 bytes.

- The performance of memory with respect to a write and read operation of 1 byte, 10 bytes and 100 bytes would be different.
- When requested for 1 byte it would need row and column address to access the 1-byte memory, similarly, to access 10 bytes of continuous memory we would need column address and row address the overhead for read and write operation would be same as the one byte read. The timing required won't exceed the time required for 1 byte read.
- Whereas for reading or writing 100 byte we only need to specify row address to read an entire page, so the overhead is much lesser as compared to 1 byte and 10 bytes read.

Review 18.3 What are the primary signals that comprise the EIA-232 interface standard? Give a brief description of each and its function in a data exchange.

The standard specifies signals for 22 of the available pins. The full standard specifies that the DTE device uses a 25-pin DB25P (male) connector and the DCE end uses a mating 25 pin DB25S (female) connector.

TXD. Data transmission line from DTE to DCE.

RXD. Data transmission line from DCE to DTE.

DSR. Data Set Ready from DCE to DTE – intended to inform the DTE that the data set has a valid connection, has completed whatever initialization might be necessary, and is ready to engage in a message exchange. If the connection drops during the exchange, this signal is de-asserted.

DTR.Data Terminal Ready from DTE to DCE – intended to inform the DCE that the data terminal has completed whatever initialization might be necessary, is ready to engage in a message exchange, and would like to open a communication channel.

RTS. Request to Send from DTE to DCE – intended to inform the DCE that the DTE had data to send and for the DCE to do whatever was necessary (dial, ensure carrier, ensure a line, etc.) to affect the communication.

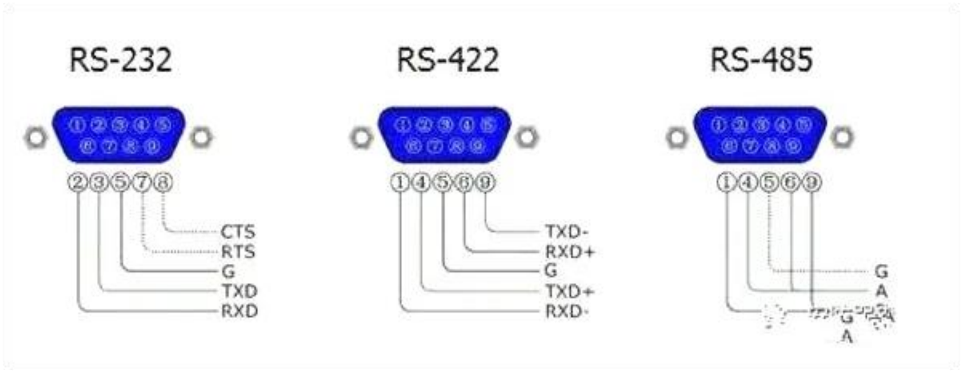
CTS. Clear to Send from DCE to DTE – intended to inform the DTE that the DCE had done its job and data could now be sent.

CD. Carrier detect – intended to indicate that a connection has been established and an answer tone has been received from the remote modem.

SG. Signal ground – the name says it.

Problem 18.1 The chapter discusses four different, commonly used, networked architectures. The Electronic Industries Association (EIA) specifies a number of others that can be found as part of various embedded applications. Three of these include the EIA-422, 423, and 485 architectures.

How are these different from the four models discussed in this chapter?



- Chapter 18 discusses EIA 232, UART, I2C, CAN network architectures.
- EIA-422, 423 and 485 are physical layer serial communication protocols. EIA 232 is serial communication protocol as well.
- However, these are different than EIA 232 in terms of cable length, data rates and driver Output voltage levels.
- EIA 422, 423 and 485 are similar to the models given in the book, benefit of using a differential signal as opposed to a single ended signal is that the system has better immunity to noise allowing for longer cable lengths.
- Except EIA 423, EIA 422 and 485 are faster compared to the protocols given in the book.
- EIA 232 is single receiver and EIA 422, 423 and 485 are multi receiver protocol.
- Compared to I2C, EIA-422, 423 and 485 are longer distance and asynchronous protocols.

Where might such models be used in an embedded application?

- RS 232 system can have 1 device transmitting at a time with 10 devices receiving in a system. RS422 systems are often used in industrial controllers.
- The BBC Micro computer used RS-423 with a 5-pin DIN connector. DEC used it extensively with a Modified modular Jack connector. This was sometimes called "DEC-423"
- RS485 systems can have 32 devices transmitting in a system with 32 devices receiving. RS485 systems are often used in CCTV applications to control the Pan, Tilt and zoom functions, telecom infrastructure, high-speed data links, and low-voltage microcontroller communications.

Ref:

https://www.st.com/en/interfaces-and-transceivers/rs-422-rs-423-rs-485.html

https://en.wikipedia.org/wiki/RS-423

Appendix I

```
#include "mbed.h"

uint32_t sat_add_U8(uint32_t a, uint32_t b)
{
   return (a > 0xFF - b) ? 0xFF : a + b;
}

uint32_t sat_add_U16(uint32_t a, uint32_t b)
{
   return (a > 0xFFFF - b) ? 0xFFFF : a + b;
```

```
}

uint32_t sat_add_U32(uint32_t a, uint32_t b)
{
   return (a > 0xFFFFFFFF - b) ? 0xFFFFFFFF : a + b;
}


uint32_t sat_mul_U8(uint32_t a, uint32_t b) {
   if(b) return (a > 0xFF / b) ? 0xFF : a * b;
   return 0;
}

uint32_t sat_mul_U16(uint32_t a, uint32_t b) {
   if(b) return (a > 0xFFFF / b) ? 0xFFFF : a * b;
   return 0;
}

uint32_t sat_mul_U32(uint32_t a, uint32_t b) {
   if(b) return (a > 0xFFFFFFFF / b) ? 0xFFFFFFFF : a * b;
   return 0;
}


int32_t sat_sub_8(int32_t a, int32_t b)
{
   return(a-b < 0x7F)? ((a-b<0x80)? a-b :0x80): 0x7F;
}

int32_t sat_sub_16(int32_t a, int32_t b)
{
   return(a-b < 0x7FFF)? ((a-b<0x8000)? a-b :0x8000): 0x7FFF;
}

int32_t sat_sub_32(int32_t a, int32_t b)
{
   return(a-b < 0x7FFFFFF)? ((a-b<0x8000000)? a-b :0x8000000): 0x7FFFFFF;
}



int main() {
   uint32_t ans1, ans2, ans3, ans4, ans5, ans6, ans7, ans8, ans9, ans10;
   uint32_t  ans11, ans12, ans13, ans14, ans15, ans16, ans17, ans18, ans19;
```

```
    int32_t ans20, ans21, ans22, ans23, ans24, ans25, ans26;

    ans1 = sat_add_U8(0x80, 0x80);
    ans2 = sat_add_U8(0x00, 0xFF);
    ans3 = sat_add_U8(0xFF, 0xFF);
    ans4 = sat_add_U8(0xF1, 0x0F);

    ans5 = sat_add_U16(0x8000, 0x8000);
    ans6 = sat_add_U16(0x000F, 0xFFF0);
    ans7 = sat_add_U16(0xFFFF, 0xFFFF);
    ans8 = sat_add_U16(0xFFF1, 0x000F);

    ans9 = sat_add_U32(0x80000000,0x80000000);
    ans10 = sat_add_U32(0xFFFFFFF1,0x0000000F);

    printf("addition tests:\n\r1. %u,\n\r2. %u,\n\r3. %u,\n\r4. %u,\n\r5. %u,\n\r6. %u,\n\r7. %u,\n\r8.
%u,\n\r9. %u,\n\r10. %u\n\r", ans1, ans2, ans3, ans4, ans5, ans6, ans7, ans8, ans9, ans10);

    ans11 = sat_mul_U8(0xFF, 0x02);
    ans12 = sat_mul_U8(0xFF, 0x01);
    ans13 = sat_mul_U8(0x0F, 0x0F);

    ans14 = sat_mul_U16(0xFFFF, 0x0002);
    ans15 = sat_mul_U16(0xFFFF, 0x0001);
    ans16 = sat_mul_U16(0x00FF, 0x00FF);

    ans17 = sat_mul_U32(0xFFFFFFFF , 0x00000002);
    ans18 = sat_mul_U32(0xFFFFFFFF , 0x00000001);
    ans19 = sat_mul_U32(0x0000FFFF , 0x0000FFFF);

    printf("multiplication tests:\n\r11. %u,\n\r12. %u,\n\r13. %u,\n\r14. %u,\n\r15. %u,\n\r16.
%u,\n\r17. %u,\n\r18. %u,\n\r19. %u,\n\r", ans11, ans12, ans13, ans14, ans15, ans16, ans17, ans18,
ans19);

    ans20 = sat_sub_8(127, 128);
    ans21 = sat_sub_8(127, 127);
    ans22 = sat_sub_8(-127, -127);
    ans23 = sat_sub_8(127, -128);

    ans24 = sat_sub_16(32767, 32768);
    ans25 = sat_sub_16(-32767, -32767);
    ans26 = sat_sub_16(32767, -32768);
    printf("subtraction tests:\n\r20. %d,\n\r21. %d,\n\r22. %d,\n\r23. %d,\n\r24. %d,\n\r25. %d,\n\r26.
%d,\n\r", ans20, ans21, ans22, ans23, ans24, ans25, ans26);


}
```