4060 - T29: Real Time Networking
Student Log

Andrew Marinic - 7675509

January - April 2024

Contents

1 Introduction

Intro The following is a log or perhaps a journal of my efforts in COMP 4060 - T29 - Real Time Networking. I will break the document into sections for each Month and subsections for each week. Each day will contain a paragraph explaining what I had accomplished.

2 January

2.1 January 8-12

Jan 8 I did some preliminary reading of the documents provided on CAN networks. I only spent about an hour reading, little notes.

Jan 11 Dug into the data sheets. Started examining the UART drivers we need to build. Probably 30 mins or so taken aside. Industrial Project has hit the fan a bit and taken up more time. I set up a calendar for the group as we all have fairly tight schedules right now.

2.2 January 15-19

Jan 15 We had an hour-long group meeting today. We discussed the pace of the project we would like to achieve. Send off an email to confirm some items and inform the Supervisor of our meeting availability. I started to dig into blinky again with the updated code. To my grief, I have switched to VScode and started to try and get the debugging working with the help of the other group members. I spent another hour or so getting the debugging to work in VScode, I stopped once I had the debugger open, openOCD via VScode. I started to get a little frustrated and had to finish my industrial project proposal so I stopped for the evening. I will discuss with the group for advice and continue later this week.

Jan 17 Today I spent around 4 hours deep-diving CAN and making notes on it. Then I proceeded to try and get debugging working one last time before asking for help. If I cannot accomplish this by, Friday I will reach out. I spent approx 1.5 hours on this, but more time was spent reading about it than coding.

Jan 18 Spent approximately 1 hour reading into the Curiosity Nano's CAN controller. I think I understand the requirements for setting up and how the INIT and CONFIG registers work together.

2.3 January 19-23

JAN 22 I spent about an hour and a half finishing my CAN notes which are visible in the Notes section. I have done some looking into coding the CAN controller of the curiosity nano.

- https://github.com/MikroElektronika/mikrosdk_click_v2/tree/master/clicks/canfd3 Here is the CAN FD 3 example code. I gave it a quick browse for something valuable but was mainly looking for CAN controller init code keeping it in my back pocket though
- https://microchip-mplab-harmony.github.io/reference_apps/apps/sam_e51_cnano/same51n_mikroe_click/readme.html I also gave themse projects a quick brows for the same as above, but I don't think any use the CAN controller just UART.
- https://microchip-mplab-harmony.github.io/reference_apps/apps/sam_e54_xpro/same54_can_usb_bridge/readme.html#hardware-used I do however think there is something useful here. It uses the same micro controller so it should have INIT, read/write code, ect. This will take some digging though. Today I put in approximately 3.5 hours into researching and documenting.

JAN 24 I tinkered more with debugging for about an hour or two this morning. It seems to be somewhat working but I am still not getting proper debugging and have no output from the debug calls. I will give it one last go. I think we will plan a group in person meeting, maybe input from the other students will help. I maybe just need to keep fresh eyes on it and stop getting frustrated and understand it better, but config files are frustrating to me. In a more productive note, I have started some framework for my drive code. I am going to clean up some of my I2C code for my gyro so I have a device to poll data from and send over the CAN network. This way I can make sure I understand the new systimer and clock speed a little better with something I understand. I suspect I will have to use some prescalers to compensate for the 60 times faster clock. I will do a commit at end of day and upload this as a pdf.

2.4 January 26 - February 1

JAN 29 Today I continued my efforts of attempting to get the CAN0 controller initialized. I got some preliminary coding done with it. I sort of got confused with some of the usage documentation and searched for some reference code online. I am having serious troubles finding anyone who has used the CAN controller on a SAM E5 device and the same sam.h dependency. I continued by porting over more of my I2C code. In total I worked on this for about 2.5 hours today. GitHub will have a commit by end of day along with updated pdf. VS Code configs continue to delay progress, but not majorly. We have scheduled a virtual meeting for Wednesday January 31st.

JAN 31 Today we had a group meeting. The minutes will be in the notes. Jacob helped me debug my debugger issues and it appears to be a bunch of version issues. I think its time to setup a Docker. I think Landon may have one. I spent about an hour in the meeting and and additional hour working on the silly version stuff.

3 February

3.1 February 5-9

FEB 5 I spent some time attempting Landons docker but I quickly abandoned that when I had the realization that python source has a make command to do an altinstall alongside the main install. This finally got gdb working, but my debug still does not work. It no longer hangs as it did before but I an getting no response back from the device. Perhaps my USB cable actually is bad. I will attempt a few other to see if this fixes the problem. Only about an hour spent today with no major coding. Intellisense still broken and I hate VS code.

FEB 7 I finally have all my old code ported over. I am now testing. I suspect the new sysTimer settings have complicated things. The button no longer works. I quickly re-installed the very cool named project kyber and tested the button with my old code. It works "flawlessly" in the sense of it works as it did before. Button works, gyro communicates. I have even tried directly copying and not reformatting my code, again does not work. Must investigate settings more but wont let it slow me down for now. Here is my game plan:

- Finish CAN drivers
- Get I2C get gyro data on a schedule
- Attempt to send gyro data over CAN

As far as continuing to debug my old code, I am going to attempt to set up another timer and PWM it to see exactly how fast the timers are moving. The force is with me, debug is now working, I tried some more stuff and eventually put in a new usb cable, not sure if that was it. Maybe my uhhhhh usb cable was "bad" or "crap" or other less civil synonyms. I can now remove this burden from my todo list. I can now use this for debugging my button and other code. Had group meeting. Finished up some code before I called it for the day and changed focus.

3.2 Feb 25th - 28th

FEB 25 Back into the swing of it. I have continued to follow the sample code I have followed. Today I fixed the circular reference in my I2C code. I made some circular references when I was separating my I2C code from my just gyro code. I re-merged them for now as I wanted to see if the I2C is still working with the systimer. I can't get messages like I used to in my project but I suspect its a buad rate issue. I did a small amount of tinkering to see if it was a quick fix. Unfortunately I didn't manage to send anything other than a start signal. I will tinker with this more another time. I shifted my focus to the CAN controller for an 3 hours. I followed along with the example code that Jacob had shown me a in our Jan 31st meeting. I also discussed some of the topics with Jacob on discord. I started to use the value assignment functions from sam.h as they

are significantly better at bit bashing the registers all at once. I set the quanta with small values for testing. I have also turned on the feed back loop. The data analyzer isn't showing me anything interesting yet. I have tried to use the debugger to set my base ram location for messages, but have not managed to get the same address back. Jacob and I have agreed to a discord discussion on Wednesday, we wait for Landon's response. The debugger is proving very valuable. I will pick Jacobs brain more on this ram issue if I can't figure it out before then. Today in total I worked on the project for approximately 4.5 hours.

FEB 28 We had a group meeting today: All attended. Landon appears to be having the same power issues with their board. Jacob and I provided sanity check conversation such as "No that works for me" sort of thing. Landon is trying a new bread board, and I have offered for him to test on my board Tues/Thursdays when we are on campus at the same time. The three of us quickly discussed the logistics of initializing (and how simple it is) and the logic of sending messages. After this Landon had to leave (about 45 min to an hour). We discussed Jacob not being able to get CAN+/- to work together. Jacob and I discussed some debugging methodologies. I wanted to make sure that I was setting my memory address registers (FLSAA/LLS and TX/RX fifos) correctly by writing back the value in the address registers to the debugger. We discovered that there is some de-referencing shenanigans so getting the address of the pointer requires some extra manipulation to get the address.

```
uint32_t␣*addr␣=␣&msgRam;
dbg_write_u32(&addr,1);
```

We quickly tinkered with this and discussed our values we were seeing. After this we wrapped up the meeting. (1H15M total). I now continue to develop and get my CAN initiation done hopefully. Little break for other classes at 1:00pm (ie 9:30am to 1:00pm work time). I will continue with this for a bit longer after I get some prof practises done. I actually didn't get much done after this break, I had to focus on my professional practises presentation and markin 2280. My next goal is to get a message to send over CAN. I know what I have to do for that. I also had a last minute "what if?" for Jacob's issue with CAN+/-

4 March

4.1 March 1 - 8th

MARCH 3rd I did a little research into actually sending messages before I start coding away at it. No more than 30 min or so. I also looked into if I was allocating memory correctly but need to look at their (sample code) struct they use for it. Professional practices presentation has consumed my day.

MARCH 6th I arrived at school early today for the meeting which means I will be getting more reading done than coding. I intend to code what ever I

miss at home later. I am continuing to examine the tx code in the sample code I am using. I found and looked into message filtering. I get why there are two rx fifos now. Its a nice way to be able to set separate filters and not have to change another. I have gotten some of the frame work for sending and receiving done before our meeting. I will continue to work on it tonight.

4.2 March 10th - 16th

MARCH 10TH I did some literature review today breifly. I have read section 39.9 CAN message RAM section an absurd amount. I understand how it is setup, but I am worried that my memory allocation method may be faulty and DMA may not work. I can see how it can work, but I can also see how we may have some errors. I may have spent an hour and a half.

MARCH 11th I think I finally get how the messages are being read. I understand both have headers that occupy some of the buffer. I am a little confused about some aspects. I have emailed about doing some sanity checks. I spent about 5 and a bit hours linking the fifo/buffer control registers to the use of the RAM buffer. I.e linked 39.8 and 39.9 to the transaction codes in the sample code I have been referring. I have removed some aspects of their code as they account for different data length messages, I want to send the simplest 1 byte message. I am uncertain if I am correctly moving around my pointer for the data. I.e when am I using byte addressing vs working within a word and how big some of these aspects may be. I feel like this DMA stuff would be easier in assembly that it is in C which is not something I thought I would ever say. I am going to get some simple testing code in blinky ready after today. I need to tinker with another project for a bit. I may or may not work more on this. In other news my CEL went on in my car and I had to run it with my OBDII scanner which has given me some motivation to finish CAN asap.

MARCH 15 Quick little reminder to myself about one of my Sunday/Monday jobs and goals. I added a comment to point myself in the right direction for when I start coding. I had to invigilate 2160 today so that ate up my morning. I did some more reading about the new hardware and started some notes on it. I am going to track down additional sample code in the next few days.

5 NOTES

5.1 CAN Overview

Controller Area Network (CAN) is a standard for micro-controller and device communication. It uses messages, and was originally designed as a multiplexing method. The physical layer utilises twisted pairs (CAN+ and CAN-). Messages are framed with IDs which dictate priority. Logical 0 is Dominant and logical 1 is recessive. This means IDs that have larger values (1's in high bit places) are lower priority. Arbitration is done via first bit with a 1. All nodes see transmission.

Nodes All nodes can send and receive, but not at the same time. The priority is determined by the frame ID. Messages are transmitted using non-return-to-zero (NRZ) format
All nodes require the following:

- Some controller :CPU/Microprocessor/host processor/micro controller
  - Decides what messages mean and what to transmit
  - Handles talking to other devices
- CAN Controller
  - Receiving: Stores bits until entire message is received, then can trigger interrupt for retrieval
  - Transmitting: If the host, can send messages via CAN controller in a serial manner.
- Transceiver (ISO 11898-2/3) Medium Access Unit (MAU)
  - Receiving: converts at the CAN bus level for CAN controller use, protective layer for CAN controller.
  - Transmitting: converts bit stream from CAN controller to the CAN bus

5.2 CAN: IDs and arbitration

ID Arbitration When nodes transmit they see all messages including their own. If they transmit a 1 and see 0 they quit and lose arbitration. This is because 0 is dominant and they then know they do not have priority. Because of this, whenever there is a collision the lower ID will win. When a collision does happen. The recessive message waits for the dominant message + 6-bit clocks then attempts again This means the first frame to transmit a 1 is the loser, thus highest priority id frame is all 0s followed by 00....001

IDs as priorities Using ID for the type of data, or the sending node ignores the fact ID is also used as message priority. This leads to poor real-time performance. CAN bus is limited to around 30% to ensure deadlines if you don't build around the priority. Otherwise, you can achieve 70 to 80% CAN bus usage and have reliable deadlines.

5.3 CAN: Bit timing

Nominal Bit Time: Time it takes to send bit components:

Synchronization Synchronization is important to the CAN protocol. It prevents errors and allows for arbitration's to occur. Recolonization occurs every single recessive to dominant transmission during the frame. In order to sync the nominal bit time is segmented into quanta and then certain aspects can be altered to allow for synchronization. The nominal bit time is broken down in the following way. Each are assigned a number of quanta. For example a system where we break our nominal bit time into 10 quanta.
Sync (1 quanta)
Propagation (3 quanta)
Phase segment 1 (3 quanta)
Phase segment 2 (3 quanta)
Synchronization occurs as follows.

1. **Bus Idle** -> wait for first recessive to dominant transition
2. **Hard synchronization**
3. **Resync** occurs on every recessive to dominant transition during the frame (message?)
    a. CAN controller expects this at multiple of nominal bit time.
    b. Else It adjust nominal bit time accordingly.

Resync and Adjustment process:

- Produce a number of quanta to divide the bits' segments into time slices.
  - The number of quanta can vary based on the controller
  - The quantity of quanta a segment is assigned can vary depending on system needs
- On out-of-sync (before or after) transition controller calculates the time difference, to compensate:
  - If we need to lengthen we do so to phase 1
  - If we need to reduce time we do so in phase 2.
- As a result of either a or b, we have adjusted the timing of the receiver to the transmitting node and synchronized them.

- We continuously do so at every recessive to dominant transition to keep synchronization
  - This reduces errors induced by noise (random error)
  - Allows for resync to nodes that lost arbitration back to the one that won previously.
  -

5.4 CAN Protocol Layers:

- Application layer
- Object layer
- Transfer layer
- Physical layer

We are building the transfer layer?

5.4.1 CAN Transfer layer:

- Most of the CAN standard applies to this layer, it is what receives messages from the physical layer and into the object layer for use in the application.
- Transfer layer is responsible for:
  - Synchronization
  - Bit timing
  - Message framing
  - Arbitration
  - Acknowledgement
  - Error Detection
  - Signalling
  - Confinement
- To accomplish the previous responsibilities, it performs the following tasks:
  - Fault confinement
  - Error detection
  - Message Validation
  - Arbitration
  - Message framing

- Transfer rate and timing
- Information routing

Physical layer

- Pinout:
  - Pin 2: CAN- (Low)
  - Pin 3: GND
  - Pin 7: CAN+ (High)
  - Pin 9: CAN V+ (power)

5.5 CAN Frames:

- Two types of frame format
  1. Base frame format
     i. 11- bits for identifier
     ii. IDE bit dominant
  2. Extended frame format
     i. 11- bit identifier + 18-bit extension = 29-bit identifier
     ii. IDE bit recessive
- There are four types of frames.
- Regardless of type all begin with a start of frame (SOF) bit to signal start of frame transmission.
- Frame types:
  1. Data Frame: a frame containing node data for transmission .
  2. Remote frame: requests transmission of an identifier
  3. Error frame: frame type for any node detecting an error.
  4. Overload frame: a buffer/delay for data or remote frame.

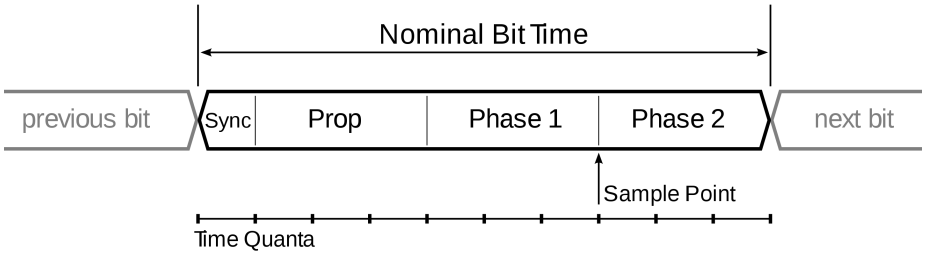


Figure 1: CAN Bit Timing: Wikipedia

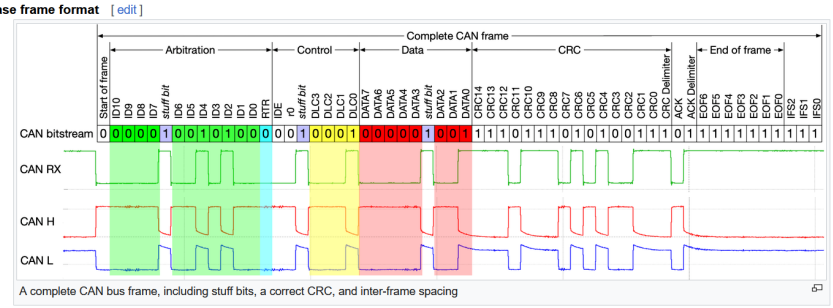
Data frame [edit]

The data frame is the only frame for actual data transmission. There are two message formats:

- Base frame format: with 11 identifier bits
- Extended frame format: with 29 identifier bits

The CAN standard requires that the implementation must accept the base frame format and may accept the extended frame format, but must tolerate the extended frame format.

Base frame format [edit]



A complete CAN bus frame, including stuff bits, a correct CRC, and inter-frame spacing

The frame format is as follows: The bit values are described for CAN-LO signal.

Field name	Length (bits)	Purpose
Start-of-frame	1	Denotes the start of frame transmission
Identifier (green)	11	A (unique) identifier which also represents the message priority
Stuff bit	1	A bit of the opposite polarity to maintain synchronisation; see § Bit stuffing
Remote transmission request (RTR) (blue)	1	Must be dominant (0) for data frames and recessive (1) for remote request frames (see Remote Frame, below)
Identifier extension bit (IDE)	1	Must be dominant (0) for base frame format with 11-bit identifiers
Reserved bit (r0)	1	Reserved bit. Must be dominant (0), but accepted as either dominant or recessive.
Data length code (DLC) (yellow)	4	Number of bytes of data (0–8 bytes)[a]
Data field (red)	0–64 (0-8 bytes)	Data to be transmitted (length in bytes dictated by DLC field)
CRC	15	Cyclic redundancy check
CRC delimiter	1	Must be recessive (1)
ACK slot	1	Transmitter sends recessive (1) and any receiver can assert a dominant (0)
ACK delimiter	1	Must be recessive (1)
End-of-frame (EOF)	7	Must be recessive (1)
Inter-frame spacing (IFS)	3	Must be recessive (1)

Figure 2: Data frame: Wikipedia

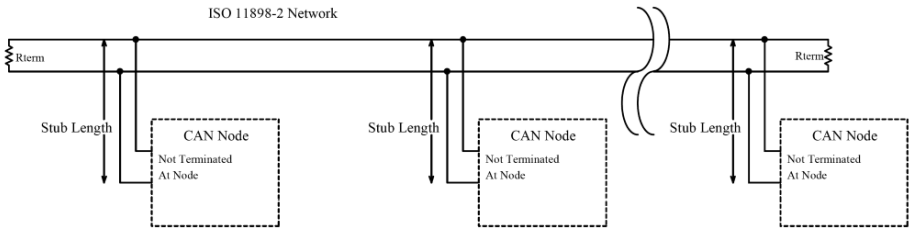


Figure 3: CAN network: https://tekeye.uk/automotive/can-bus-cable-wiring

University of Manitoba

Meeting Minutes

Date and Time	Jan 31st 2024 @ 9:00 am
Venue	Discord - Lounge
Participants	Andrew Marinic & Jacob Janzen
Absent	Landon Colburn

Minutes of the Meeting

Item	Notes and Discussion	Group
1	I need help with my debugger still	**Andrew**
2	Jacob helped me find that I was missing an outdated version of libncurses	**Group 11**
3	We tracked down, and Andrew resolved	**Group 11**
4	Discussing CAN and figuring out if INIT resets CCE sort of like the PORT Toggle.	**Andrew & Jacob**
5	Found reference material `https://github.com/majbthrd/SAMC21demoCAN`	**Jacob**
6	CAN baud rate, do we need it?	**Andrew**
7	Start at 0	**Jacob**
8	Jacob found info about how the CCE get set in the soure code on line 420 of mcan.c `https://github.com/majbthrd/SAMC21demoCAN/blob/master/mcan.c#L408`	**Jacob**
9	Conclusion that it needs to be set manaully and that INIT needs to be set to 0 and not 1 like toggle	**Jacob**
10	Have you looked at fast CAN? should we be implementing that?	**Andrew**
11	Starting with normal CAN, but agree we should look into, Click controller uses it	**Jacob**
12	Have you looked into the resistors? They should go into the CANH and CANL?	**Andrew**
13	Showed their setup and how they put in their resistors, reminded Andrew about using the +/- for CANH and CANL	**Jacob**
14	Discussion about TX/RX but neither at that step	**Andrew & Jacob**
15	Discussion about ideas for what our piece of CAN network will do Andrew wants to use Gyro Jacob wants to use RNG sfu	**Andrew & Jacob**
16	Wrap up discussion, discuss next meeting. Finished at 9:45 am	**Andrew & Jacob**

5.6 RS485-6-Click

General description Uses the THVD1429DT (transceiver from TI) half-duplex RS485 via UART, rate of up to 20 Mbps. Voltage 3.3V to 5V. I can thus be useful for multi-point implementations, with tolerance to noise and high physical distance integrity, achieved with a transient voltage suppressor. Each end should be terminated with a resistor that matches cable impedance. This parallel termination allows for the distance integrity. The total amount of BUS Nodes supported is 256. THe device has electrostatic discharge protection. Additionally it supports Electrical Fast Transient (EFT) portection which allows protection from high frequency bursts caused bu devices such as relays, switch contractors, or Heavy Duty Motors; additional surge transient protection from system power downs or short.

5.6.1 RS485-6-Click references

`https://www.mikroe.com/rs485-6-click`