Lecture 04

Lecture contents:

1-RS-232C

2-UART

3-USB

“Slides 38:58”

احنا قبل الميدترم كنا بنتكلم عن ال parallel bus واللي اخدنا عليه example مهم اللي هو ال AMBA bus اللي كان فيه AHB & APB

دلوقتي بقى هنتكلم عن ال serial buses .. بس ايه الفرق بقى بين ال serial & parallel buses ؟؟

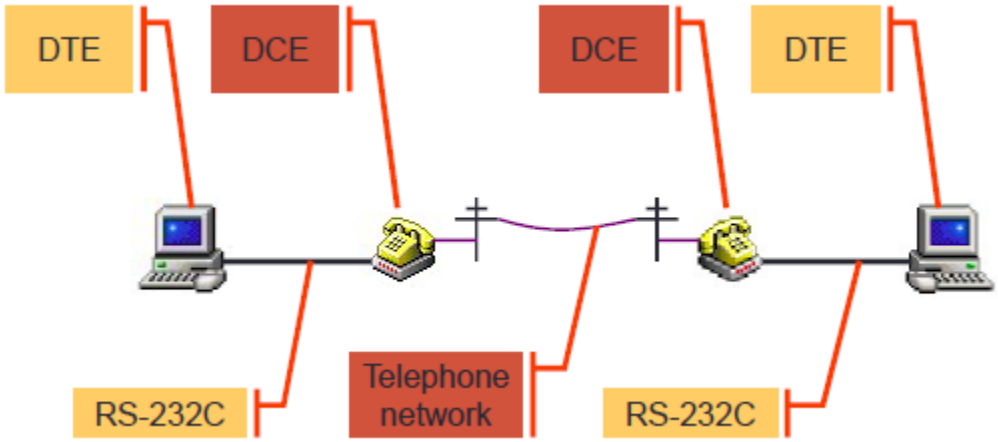
قلنا قبل الميدترم ان ال parallel bus الداتا فيه بتطلع على اسلاك كتير .. عشان نبعت 8 bits يبقى محتاجين 8 اسلاك عشان كل bit تطلع على سلكة وتوصل في نفس الوقت، بس قولنا انها في الحقيقة ممكن ماتوصلش كلها في نفس الوقت عشان ال skew وطبيعة كل سلكة

في ال serial bus بقى احنا مش عندنا غير سلكة واحدة بس ال 8 bits هيتبعتوا عليها bit by bit .. ف دي هتبقى احسن في الهاردوير وفي نفس الوقت احتمال ان يحصل interference في الداتا يبقى اقل

هنتكلم بقى عن اول example لل serial buses :

RS-232C

دة protocol قديم اتعمل في الستينات .. كان الهدف منه انه يوصل بين ال Data Terminal Equipment (DTE) "زي الكمبيوتر او ال printer مثلاً" بال Data Communications Equipment (DCE) زي ال modem .. فلو بصيت على الصورة الجاية دي:



هتلاقي ان الكمبيوتر الاولاني متوصل بسلكة على علامة التليفون (دة اللي هو ال modem)، والكمبيوتر التاني متوصل زي الاولاني
وبين ال 2 modems هتلاقي فيه telephone network .. دة تبع كورس networking مالناش دعوة بيه دلوقتي فاحنا مركزين على السلكة اللي بين الكمبيوتر والموديم .. اللي هي RS-232C

زمان كان ال data rate بتاع ال RS-232 قليل اوي فكان آخرهم ينقلوا الداتا بمعدل 20kbits/sec .. وكان طول السلك كبير وبيوصل ل 15 متر

بس دلوقتي ممكن ننقل ب higher data rate (up to 1 mbit/sec) بس عشان نعمل كدة محتاجين نقصر السلك قدر الإمكان ..بس ليه نصغره؟؟! :

دلوقتي احنا عايزين higher data rate يعني عايزين ال transmitter يبعت الداتا بسرعة، يعني الوقت اللي بين كل bit بتطلع من ال transmitter وال bit اللي بعدها يبقى قليل .. الوقت دة اسمه transmission time

ولو فاكر في المحاضرة التانية لما اتكلمنا عن ال propagation time وإنه بييجي بسبب طول السلك

فلو احنا حطينا سلك طويل (high propagation time) وسرعنا الداتا (lower transmission time) .. لو وصلنا بقى لمرحلة فيها ال propagation time اكبر من ال transmission time يبقى ال bit الجديدة هتطلع من ال transmitter على ال bus قبل ما ال bit القديمة تروح لل receiver فهيحصل interference بينهم والداتا مش هتوصل صح لل receiver

عشان نـ configure ال RS-232 نقدر نعمله serial او

point to point : يعني transmitter واحد بس بيكلم receiver واحد بس

هنركز دلوقتي على ال serial data transmission .. **ودة in general مش لل RS-232 بس** ..

احنا نقدر نبعت serial data ب 2 modes :

- Synchronous
- Asynchronous

**لو هنبعت synchronous data فاحنا عندنا طريقتين نبعت بيهم:

- <u>Pattern of the clock</u>

الطريقة دي ال transmitter بيبعت sample او pattern من ال clock بتاعته لل receiver عشان ال receiver يظبط سرعته او ال bit rate بتاعه على ال clock دي .. ودة اللي هنشوفه في ال **<u>USB</u>** بعد شوية

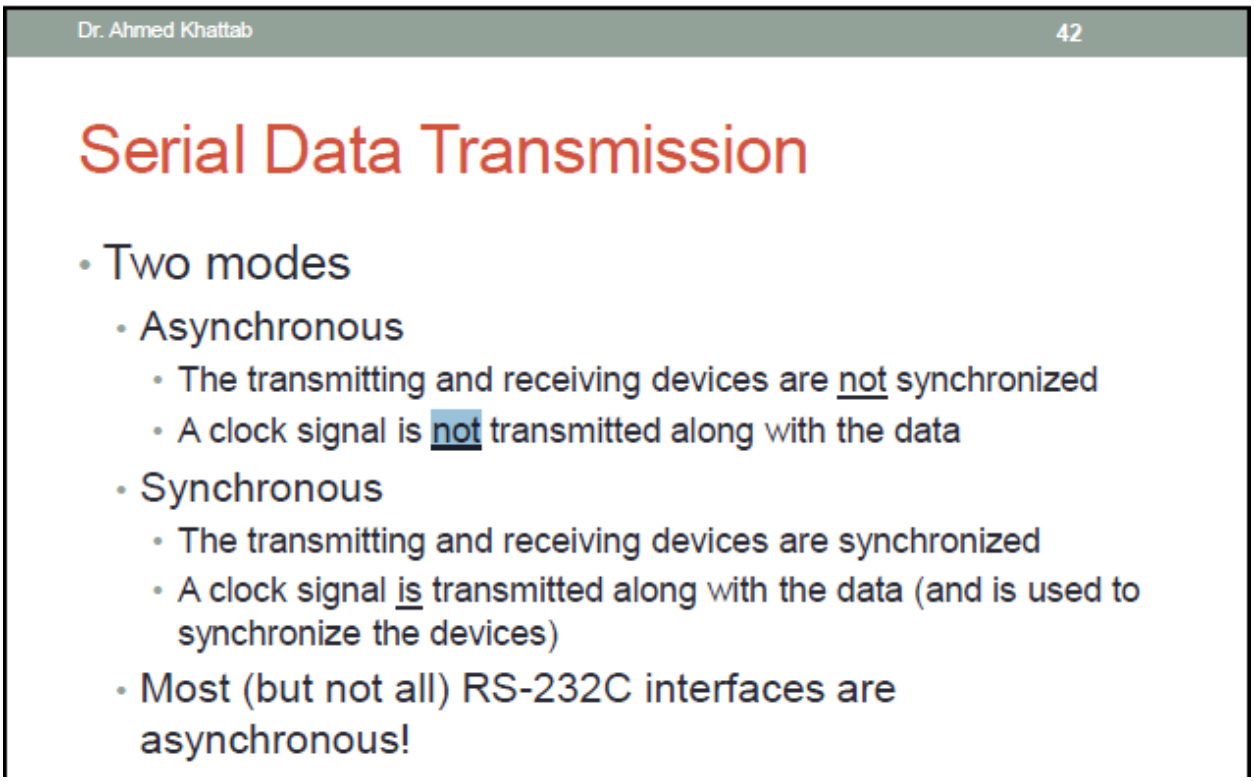
- <u>External clock to the communicating devices</u>

الطريقة دي بنجيب فيها external clock ونوصلها على كل ال devices اللي عايزينها تتكلم مع بعضها، فيبقوا كلهم ماشيين مع بعض بنفس السرعة .. ودة اللي هنشوفه في ال **<u>I2C</u>** المحاضرة الجاية او اللي بعدها ان شاء الله

**ولو هنبعت Asynchronous data اللي هو ال mode التاني فيبقى لازم ال devices اللي بت communicate كلها تكون متفقة على حاجات معينة عشان يعرفوا يتكلموا مع بعض من غير clock

ودة اللي هنشوفه دلوقتي في ال **<u>RS-232C</u>**

<u>أغلب</u> ال RS-232C بيبقوا Asynchronous ... وخلي بالك كويس من ال slide الجاية دي واللي فيها نفس الكلام اللي قلناه دلوقتي :



تعالى بقى نتكلم اكتر عن ال Asynchronous transmission

ال standard بيقول ان الداتا بتتبعت على ال transmit data line على هيئة packets .. وال packet بيبقى فيه 7 او 8 bits .. واحنا اللي بنحدد طول ال packet دة لل receiver قبل ما نبعت الداتا .. هنحدده ازاي؟؟ دة اللي هنعمله في البروجيكت ^^

وتعالى نقول ان logic 1 دة اسمه mark

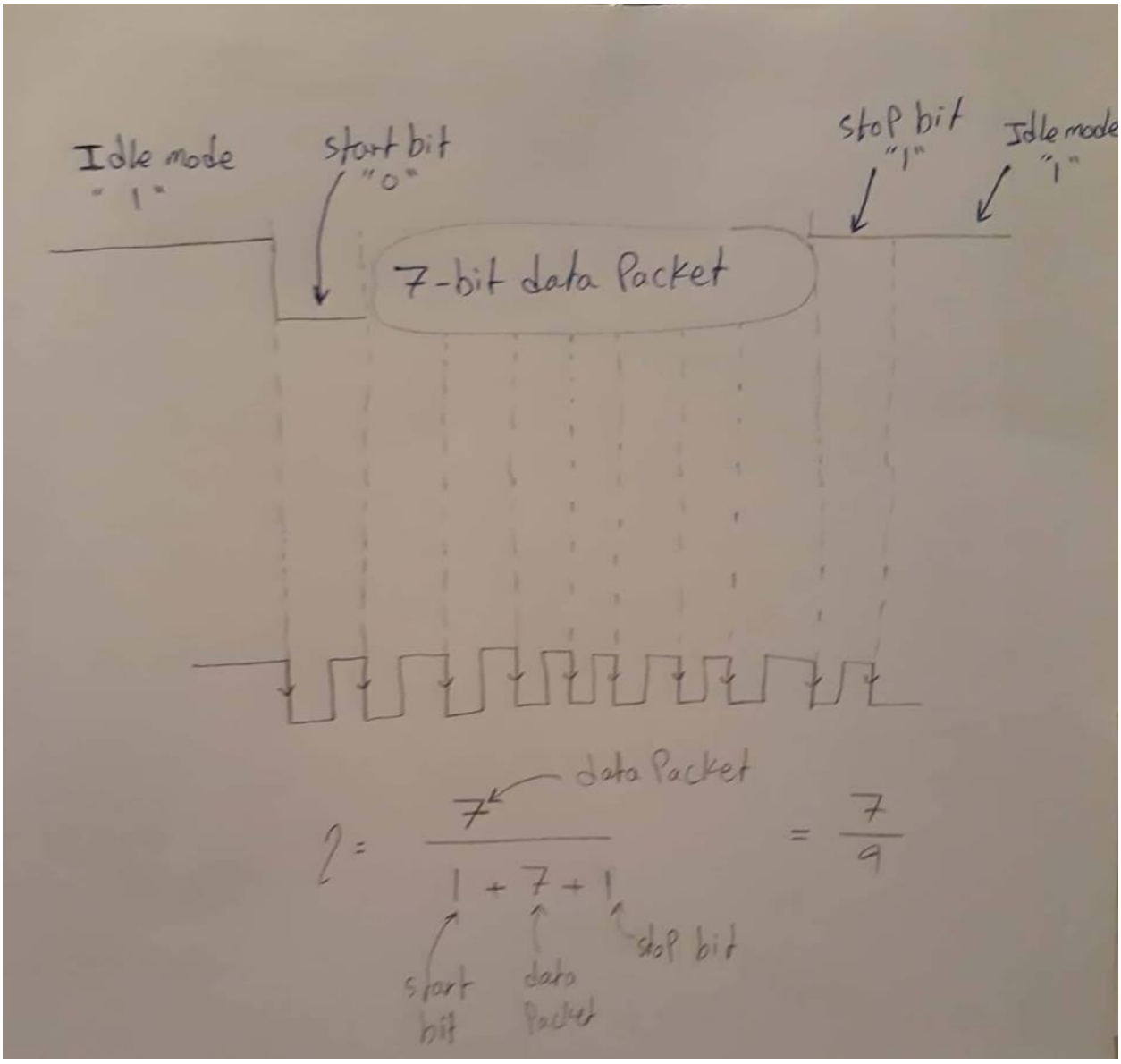
و logic 0 اسمه space

ال transmit data line دة by default بيكون في idle mode يعني عليه mark (اللي هو logic 1) طول الوقت لحد لما نبقى عايزين نبعت داتا ..

دلوقتي بقى عايزين نبعت داتا، بس ماينفعش نبعت الداتا كدة فجأة على ال bus او ال line .. فعايزين نبعت حاجة لل bus دة نقوله من خلالها استعد انا هبدأ ابعتلك داتا دلوقتي .. الحاجة دي هي ال **start bit** ... وأكيد لازم تكون مختلفة عن ال idle mode عشان ال bus يعرف يفرّق بينهم، فهتبقى logic 0 .. وبكدة ال bus هيبقى مستعد ياخد الداتا

بعد كدة هنحط الداتا على ال bus وهنعتبر اننا قايلين لل receiver اننا هنبعت 7 bits .. بعد ما نخلص هنبعت **stop bit** عشان نقول لل bus اننا خلصنا .. ال stop bit دي هتبقى mark عشان تخلي ال bus يرجع تاني لل idle mode

وهيبقى دة ال transaction اللي بيحصل :



وخلي بالك ال transmitter وال receiver بيكونوا متفقين على ال bit rate اللي هيشتغلوا بيه قبل ال data transmission زي ماكانوا متفقين انهم هيبعتوا ال packet على انه 7 bits

افرض بقى انت عايز تبعت كذا packet ورا بعضها .. ساعتها هتبعت start bit وبعد كدة ال packet الاولانية وبعد كدة stop bit .. و start bit تاني وتبعت ال packet التانية وبعد كدة stop bit تاني وهكذا ... يعني مش هتقدر تبعت start bit مرة واحدة و stop bit مرة واحدة وال packets كلها في النص بينهم زي ما كنا بنعمل في ال burst .. ليه؟؟!

لأن احنا هنا شغالين asynchronous فمع اول bit بتتبعت من ال packet ال bit rate بيبدأ يتغير بنسبة صغيرة اوي، ومع تاني bit بيتغير بنسبة صغيرة .. وهكذا لحد لما نوصل لآخر ال packet فبيبقى ال bit rate متغير بنسبة تبدأ تبقى محسوسة .. فلو احنا دخلنا كذا packet مع بعض ممكن نلاقي الدنيا باظت في النص لأن ال bit rate اتغير جامد فالداتا حصلها corruption

عشان كدة لازم هنا نبعت 1 start bit + 1 data packet + 1 stop bit + 1 start bit + 1 data packet + 1 stop bit + ….

ودة عشان كل شوية نرجع نظبط ال bit rate

فدايماً لازم أي frame يبدأ ب start bit وينتهي ب stop bit

ممكن نزود حاجة كمان في ال frame اسمها parity bit (وممكن مانزودهاش براحتنا) .. دي بنحطها عشان ن detect لو فيه error حصل في 1 bit

لو فاكر ال parity bit زمان كنا بنقول اننا لو شغالين ب even parity يبقى عدد ال logic 1’s اللي في ال packet دة even .. ولو odd parity يبقى عدد ال logic 1’s دة odd

فال transmitter هيتفق مع ال receiver هم هيشتغلوا ب parity bit ولا لأ .. ولو هيشتغلوا هيتفقوا هي هتبقى even or odd parity قبل ما يبدأوا ال data transmission .. وال receiver هيشوف لو مثلا كانوا متفقين على even parity يبقى هيشوف عدد ال logic 1’s اللي جاله لو even يبقى تمام ولو odd يبقى حصل error في حاجة معينة فهيرمي الداتا دي وهيقول لل transmitter يبعتها تاني من جديد

بنحط ال parity bit في الآخر قبل ال stop bit .. مش بنحطها في الأول .. ليه؟! :

عشان نكسب وقت .. كدة كدة ال bit الاولانية هتخرج على ال bus من ال transmitter .. ف وهي خارجة هتروح بالمرة على xor .. وبعد كدة ال bit التانية تخرج على ال bus وبالمرة تروح على

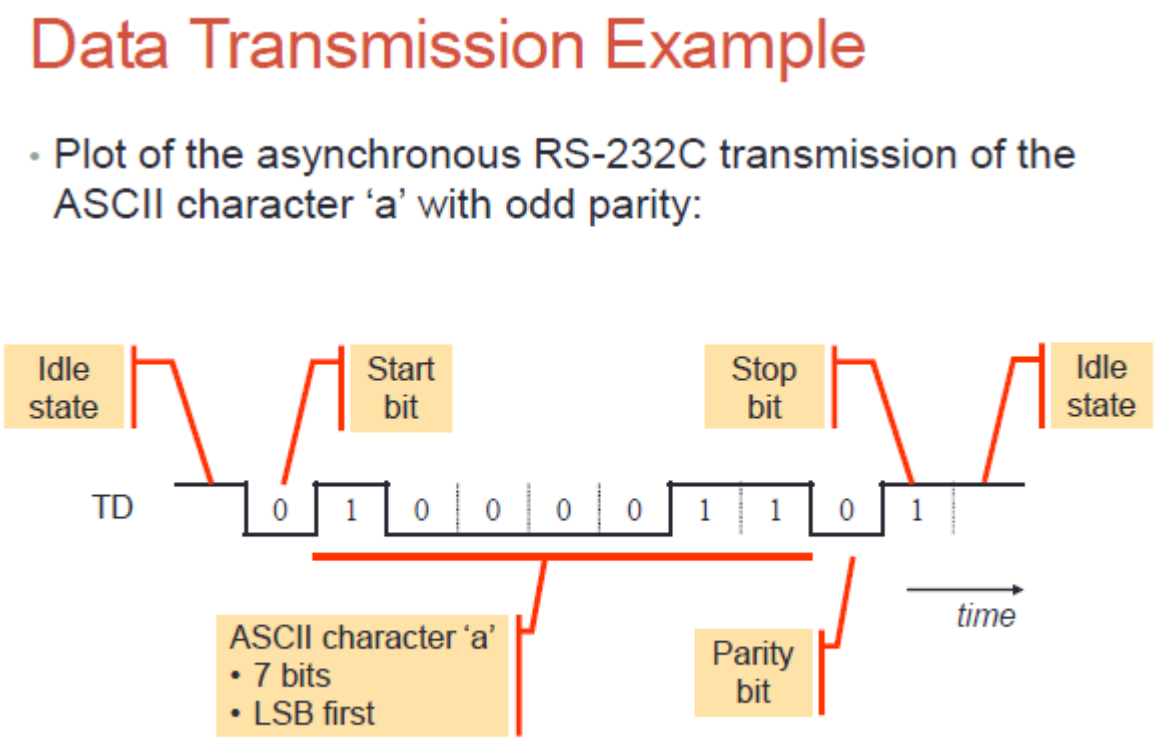
xor برضه .. والتالتة تخرج لل bus وتروح على xor مع النتيجة اللي طلعت من ال xor بتاعت اول 2 bits وهكذا لحد لما نوصل لآخر bit وتبقى النتيجة الأخيرة دي هي ال parity bit .. فال parity bit خرجت بعد اخر bit من الداتا

-احنا قلنا ان ال parity bit دي بت detect ال error لو حصل في 1 bit بس .. طب لو حصل في 2 bits ؟؟

ساعتها ال error detection هيـ fail لأن ال receiver مش هيحس بتغيير ... بس احتمال ان دة يحصل مش كبير لأن أصلا احتمال ان يحصل error في just 1 bit في ال serial buses قليل .. (فاكر لما قلنا ان ال error في ال serial نسبة حدوثه أقل لأن مافيش اسلاك كتير فمافيش skew بيحصل؟)

تعالى نجرب نبعت ال ASCII code بتاع حرف 'a' .. اللي هو 1100001 وهنبعت ب odd parity :

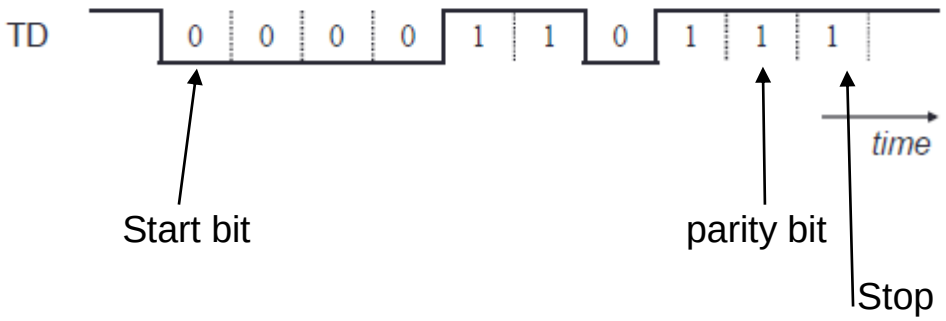
خلي بالك ان اللي بيتبعت الأول هو ال LSB مش ال MSB



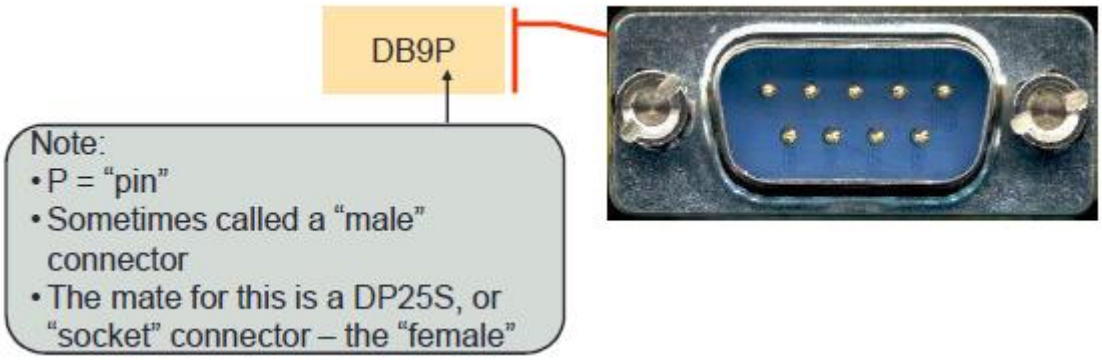
هنا ال parity bit ب 0 لأن عدد ال marks في ال **PACKET** أصلا odd (3 marks)

ولو عايزين نبعت حرف ال X :

- Plot the transmission of the ASCII character “X” over an asynchronous RS-232C channel with 7 data bits and even parity



زمان كان الكابل بتاع ال RS-232 فيه 25 pin .. بس دلوقتي بقوا 9 بس وبقى دة شكله:



ال 9 دول فيه 8 pins منهم DC والتاسعة هي اللي بتنقل الداتا

بس فيه مشكلة كبيرة هنا .. ال computer ال bus بتاعه بيبقى parallel bus .. فال 7 bits بيخرجوا كلهم من الكمبيوتر مع بعض في نفس الوقت .. ازاي بقى سلكة واحدة serial هتاخد داتا جاية من 7 اسلاك فيهم parallel data ؟؟

حل المشكلة دي موجود في ال Universal Asynchronous Receiver Transmitter (UART)

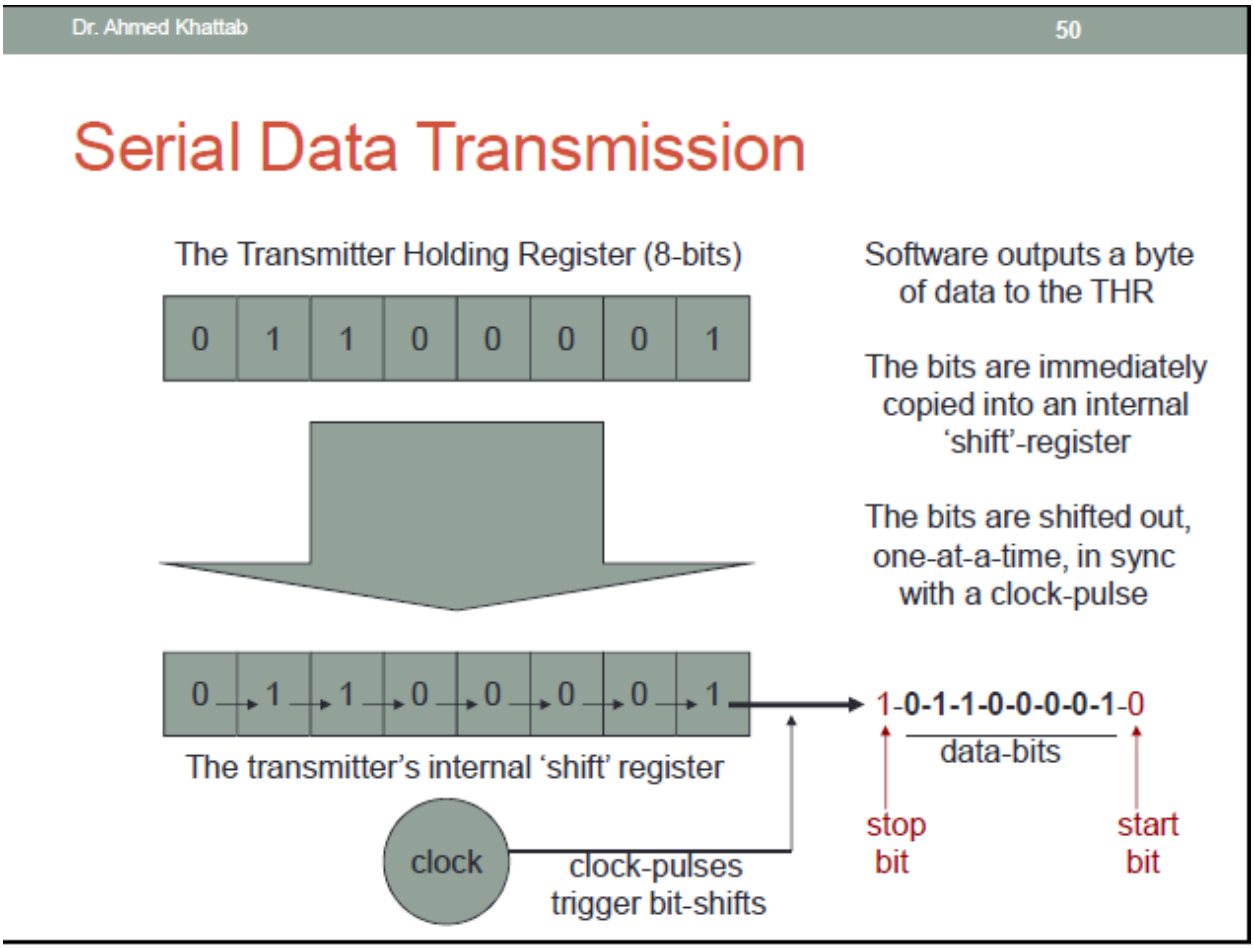
UART:

ال UART دة hardware peripheral .. والسوفتوير هو اللي بيتحكم في ال registers بتاعته

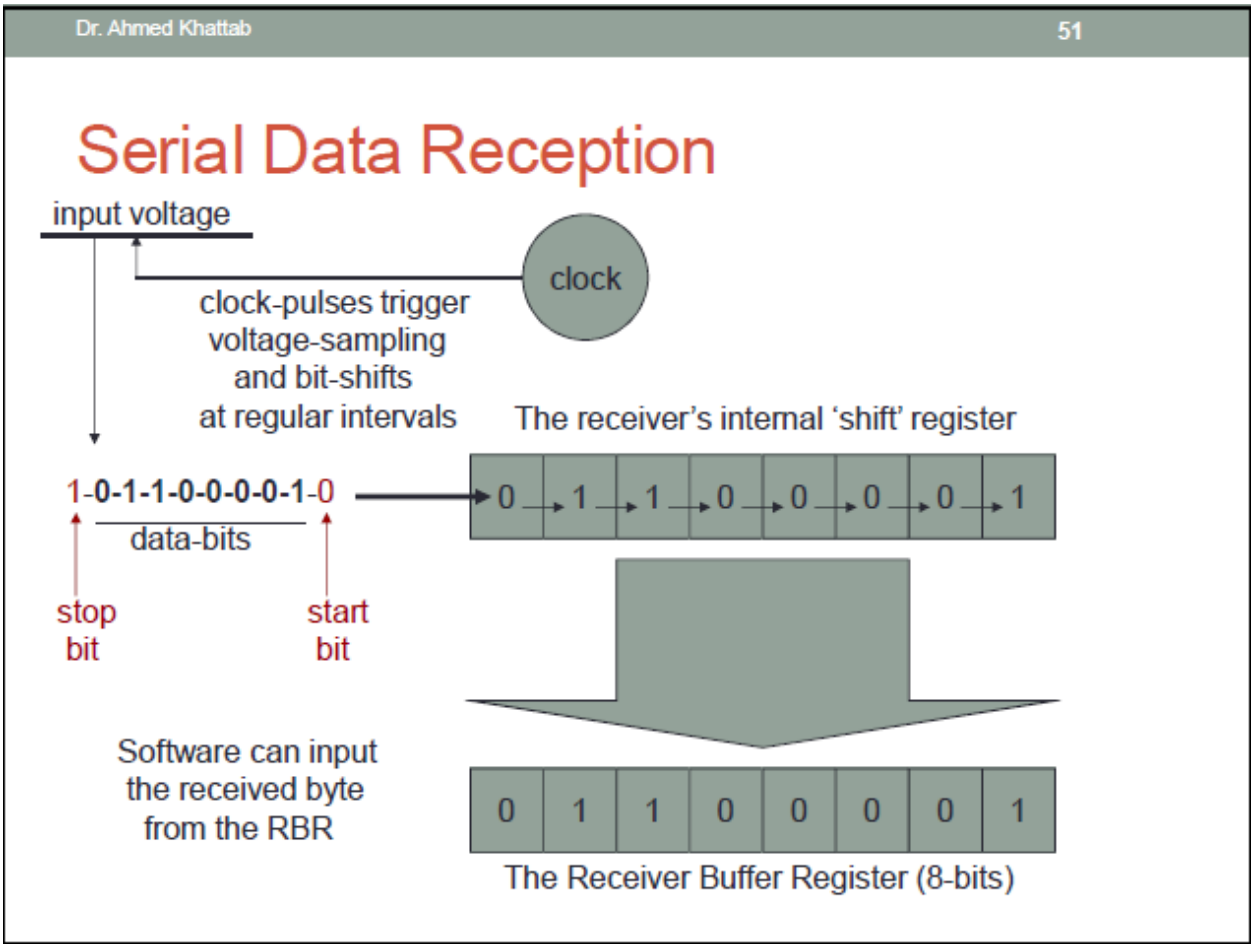
المشكلة كانت ان ال parallel data مش عارفين نخليها serial .. ففي ال UART لو احنا عايزين ن transmit data .. ال parallel data دي هتدخل على register عادي هنسميه Transmit Holding register .. بعد كدة الداتا اللي في ال holding register دي هيتعمل لها copy وتروح على shift register .. أول ما تتحط على ال shift register على طول ال transmitter هيبعت ال start bit و وال shift register هيبدأ يبعت الداتا دي bit by bit، هيبدأ بال LSB وكل bit هتتبعت في 1 clk cycle وفي نفس الوقت هتدخل على xor عشان لما يخلص ال data packet كلها يبقى ال peripheral عارف نتيجة ال xor ويحط ال parity bit (لو احنا عايزين parity) وبعد كدة يحط ال stop bit

***خلي بالك ال start & stop bits مش بيكونوا موجودين في ال shift register لأن دة protocol بيحصل دايماً

وال slide دي ملخصة الدنيا :



ال receiver بقى هيعمل نفس الموضوع بس بالعكس .. ال frame هيبدأ يجيله .. في الأول هيقرا ال start bit و هيرميها (مش هيحطها في ال shift register لأنها مش داتا) وبعد كدة الداتا هتبدأ تجيله وهيبدأ يخزنها في ال shift register لحد لما ال stop bit تجيله ... بس لو كان جاله parity bit قبل ال stop bit هيـ check عليها .. لو تمام يبقى خلاص مافيش مشاكل والداتا هتروح على ال layers اللي فوق (هتروح على ال Receiver buffer register وبعد كدة يتعمل عليها ال processing المطلوب)، بس لو مش تمام يبقى حصل مشكلة في ال transmission فهيمسح كل اللي هو عمله في ال shift register ويخلي ال transmitter يبعت الداتا مرة تانية



خلي بالك .. مش معنى ان ال UART دة asynchronous يبقى هو كدة مالوش clock ... لأ طبعا أي peripheral له clock حتى لو .asynch بس الفكرة كلها ان ال clock مش بيبقى ليها تأثير على ال transmission

كدة احنا خلصنا ال RS-232C وال UART

ندخل بقى على ال USB

USB

زمان كان كل device بيتوصل في الكمبيوتر ب bus معين وبالتالي ب protocol معين .. فكنا بنشوف مكان ال mouse ماينفعش احط فيه السلكة بتاعت ال keyboard

فالناس بتاعت Microsoft و intel و Apple والناس الكُبارة اللي زيهم قعدوا قعدة عرب كدة وقالوا يا رجالة الكلام دة ماينفعش احنا عايزين نعمل protocol يبقى universal او مُوحد لل devices وطلعوا بال USB

ودلوقتي احنا بنستخدم ال USB في كل حاجة تقريبا حتى في ال power transfer فكمان بقينا بنشحن الموبايلات عن طريقه .. طبعا فيه protocols تانية قوية زي ال I2C وال CAN & LIN بتوع ال automotive بس بالنسبة لل user العاديين هم مش بيستخدموا غير ال USB

ال USB دة synchronous يعني بيشتغل بال clock .. ولو فاكر في اول المحاضرة لما قلنا انه بيبعت pattern من ال clock في الأول عشان ال Transmitter وال receiver يتفقوا من خلاله على السرعة بتاعتهم .. هنتكلم عن ال pattern دة بعد شوية

المهم ال usb دة في الأول كان اقصى سرعة له هي 12 Mbps ودة كان USB1 .. بعد كدة ظهر حاجة اسرع اللي هي USB2 ودلوقتي معانا USB3

Universal Serial Bus (USB)

- Universal Serial Bus is a synchronous serial protocol for low to medium speed data transmission
  - USB1:
    - Full speed signaling 12 Mbps
    - Low Speed signaling 1.5 Mbps
  - USB2
    - Targets maximum signaling of 480 Mbps (effective 35 Mbps)
  - USB3
    - Targets maximum signaling of 5 Gbps (effective 625 Mbps)
- Intended devices are keyboards, mice, joysticks, speakers; other low to medium speed I/O devices

في USB2 & USB3 تفتكر الفرق اللي بين ال maximum signaling وال actual (effective) دة بيروح فين؟؟:

الفرق دة بيروح في ال overhead .. فال maximum دة معناه سرعة ال USB لو مافيش أي overhead موجود

طبعاً الفرق كبير فعشان كدة فيه option لل transmitter انه يبعت كذا package بحيث انه يقلل ال overhead دة

تعالى نسمي ال master باسم تاني (Host) وال slave هنسميه (function)

**دايماً ال USB بيبقى single master (وعشان كدة ساعات بنستخدم I2C مكانه عشان بيـ support multi masters) .. الماستر دة بيبقى متوصل ب HUB (زي مشترك كدة) وال HUB دة بيبقى خارج منه كذا slave او function

فمثلاً في الكمبيوتر او اللاب بيبقى عندنا 3 او 4 usb ports .. كلهم متوصلين من جوة بال HUB وال HUB دة متوصل بالماستر

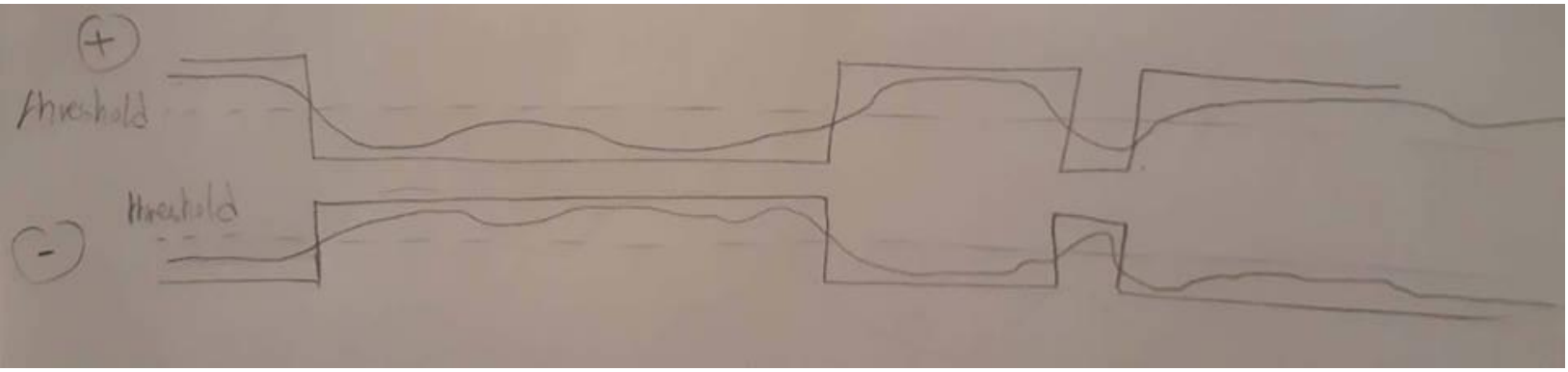
فبالطريقة دي الماستر يقدر يـ host up to 128 functions(slaves) بس مش بنعمل كدة عشان كل لما عدد ال slaves يزيد كل لما سرعة ال USB تقل

دايماً ال USB بيشتغل polling .. مافيش interrupts خالص هنا .. أصل تخيل ان انت عندك 128 device وكل واحد فيهم له interrupt بيوقف ال processor عن حاجة مهمة كان بيعملها .. عشان كدة دايماً ال processor بيروح يـ check كل شوية على ال usb devices ويشوف لو فيه حاجة جديدة حصلت فيه يروح يعمل عليها ال action المطلوب وهكذا فمش بيتعطل ويسيب كل اللي وراه، هو دة ال polling

فيه ميزة تانية حلوة اوي في ال USB هي ال hot plugging and unplugging (او plug and play) .. زمان كنا بنحتاج أسطوانات تعريفات وليلة كبيرة عشان نعمل setup لل driver بتاع ال device، بس بال plug and play الكمبيوتر بيقدر ي detect ان فيه device جديد اتوصل عليه وبيدور على ال driver بتاعه automatically ويسطبه

ندخل في الجد بقى .. دلوقتي احنا لو قطعنا أي سلكة usb هنلاقي 4 اسلاك .. واحدة V_{cc} وواحدة ground و2 differential data

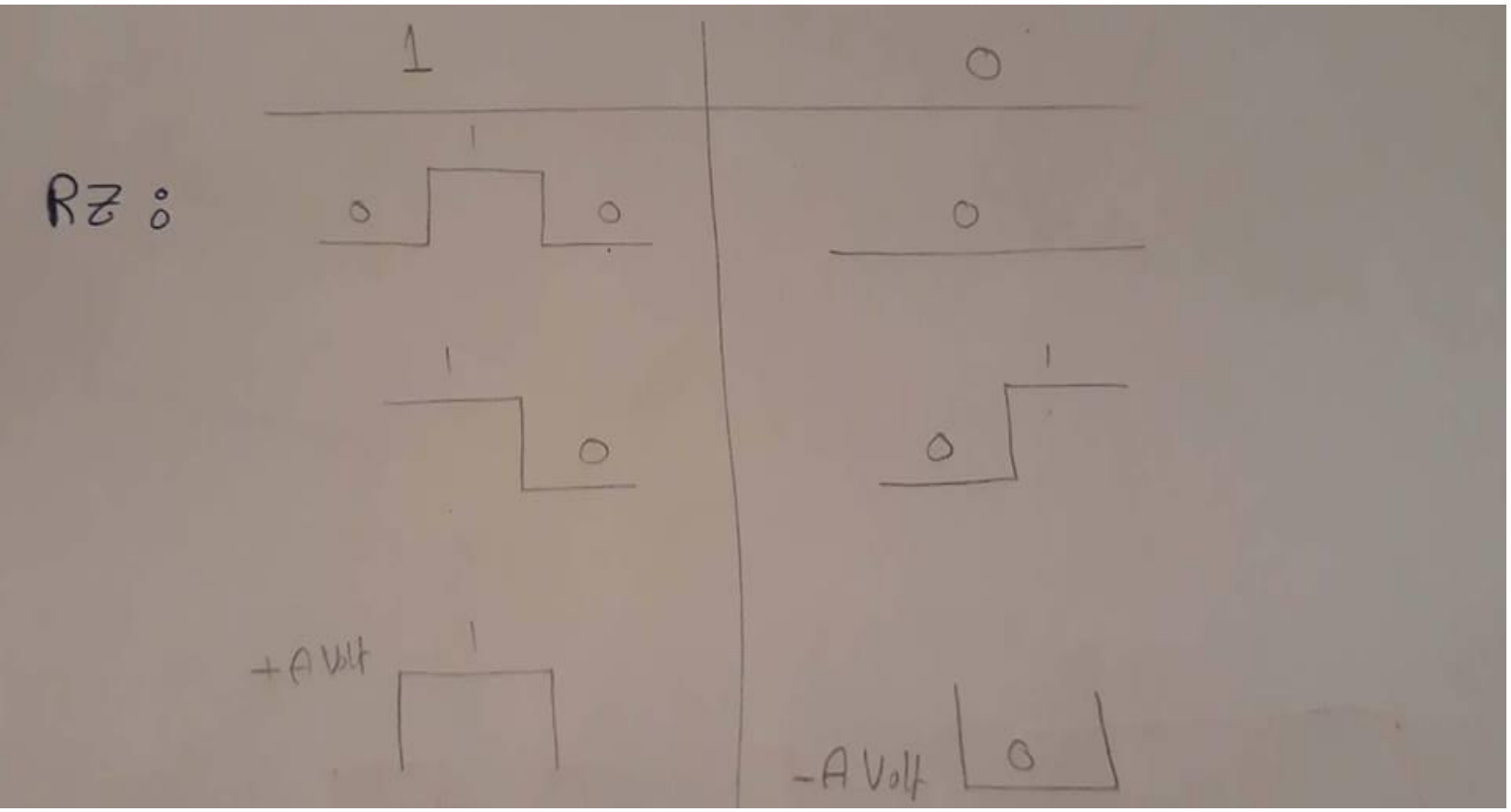
فدة معناه ان ال USB بيشتغل ب differential signaling ودة لأن ال 1 وال 0 مش بيتبعتوا & 0 5 volt بالظبط فلو ال noise خلت ال signal تعدي ال threshold ال 1 هيبقى 0 او العكس .. فال differential signal عشان ال USB يبقى more noise immune



وكمان ال USB دة half duplex protocol يعني لازم ال device يا اما يستقبل يا إما يبعت الداتا في المرة الواحدة .. ماينفعش يستقبل ويبعت في نفس الوقت

ازاي بقى بنبعت الداتا على ال USB ؟؟ :

بنبعتها عن طريق ال line code .. ولو فاكر من دكتور منى رياض ان ال line code دة كود او شكل بنعرف منه شكل ال 1 وال 0 .. فال 1 بيتبعت بشكل معين وال 0 بيتبعت بشكل تاني .. فمثلاً ممكن نبعت الداتا بطريقة من الطرق اللي جاية دي وممكن نبعتها بطرق غيرها



أول طريقة مرسومة دي اسمها Return to Zero (RZ) ... يعني لو عايز ابعت 1 .. ببعت 1 شوية (اللي هو 5 فولت أو ال high voltage) (لمدة معينة من ال clock period) وبعد كدة بنزل ل 0 في نفس ال period (اللي هو 0 فولت)

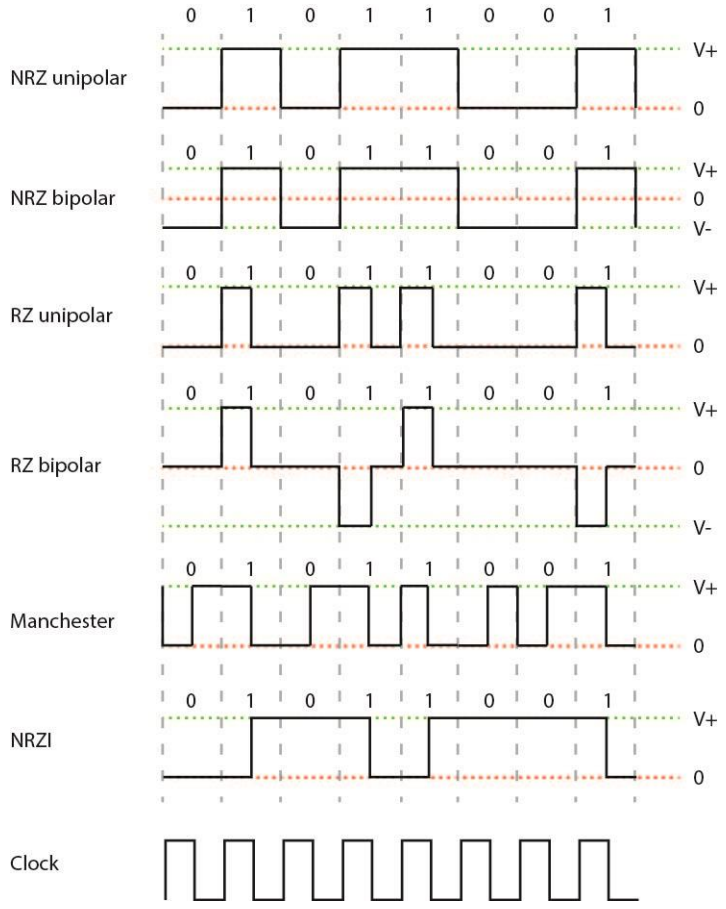
فيه بقى حاجة تانية اسمها Non Returning to Zero (NRZ) .. دي لو عايز ابعت 1 .. مش بترجع للزيرو زي ال RZ، بتفضل 1 على طول ال period

وسواء ال RZ او ال NRZ فكل واحد فيهم ممكن يبقى uni-polar او bipolar .. بس احنا مركزين في ال unipolar دايماً، الدكتور ماقالش بس غير ان ال bipolar ممكن ينزل ل amplitude = -A volt

احنا بقى في ال USB لا بنستخدم RZ ولا بنستخدم NRZ D:

احنا بنستخدم حاجة مشتقة من ال NRZ اسمها Non Returning to Zero Inverted (NRZI)

انت اكيد متلخبط دلوقتي فقبل ما نشرح ايه ال NRZI تعالى نشوف الفرق اللي بين ال RZ وال NRZ :



الصورة دي من النت مش من المحاضرة بس مجمعة الدنيا يعني .. المهم احنا مش عايزين نبص غير على ال NRZ unipolar وال RZ unipolar وال clock اللي تحت خالص

في ال NRZ لما احتاجنا نبعت 1 بعتناه على طول ال clock cycle (من أول ما بدأت لحد ما خلصت)

بس في ال RZ لما جينا نبعت 1 لقينا ان ال 1 موجود في جزء من ال cycle والجزء التاني مش موجود (رجع لل zero قبل ما ال cycle تخلص)

فهو دة الفرق بين ال RZ وال NRZ

طيب ايه بقى ال NRZI ؟؟:

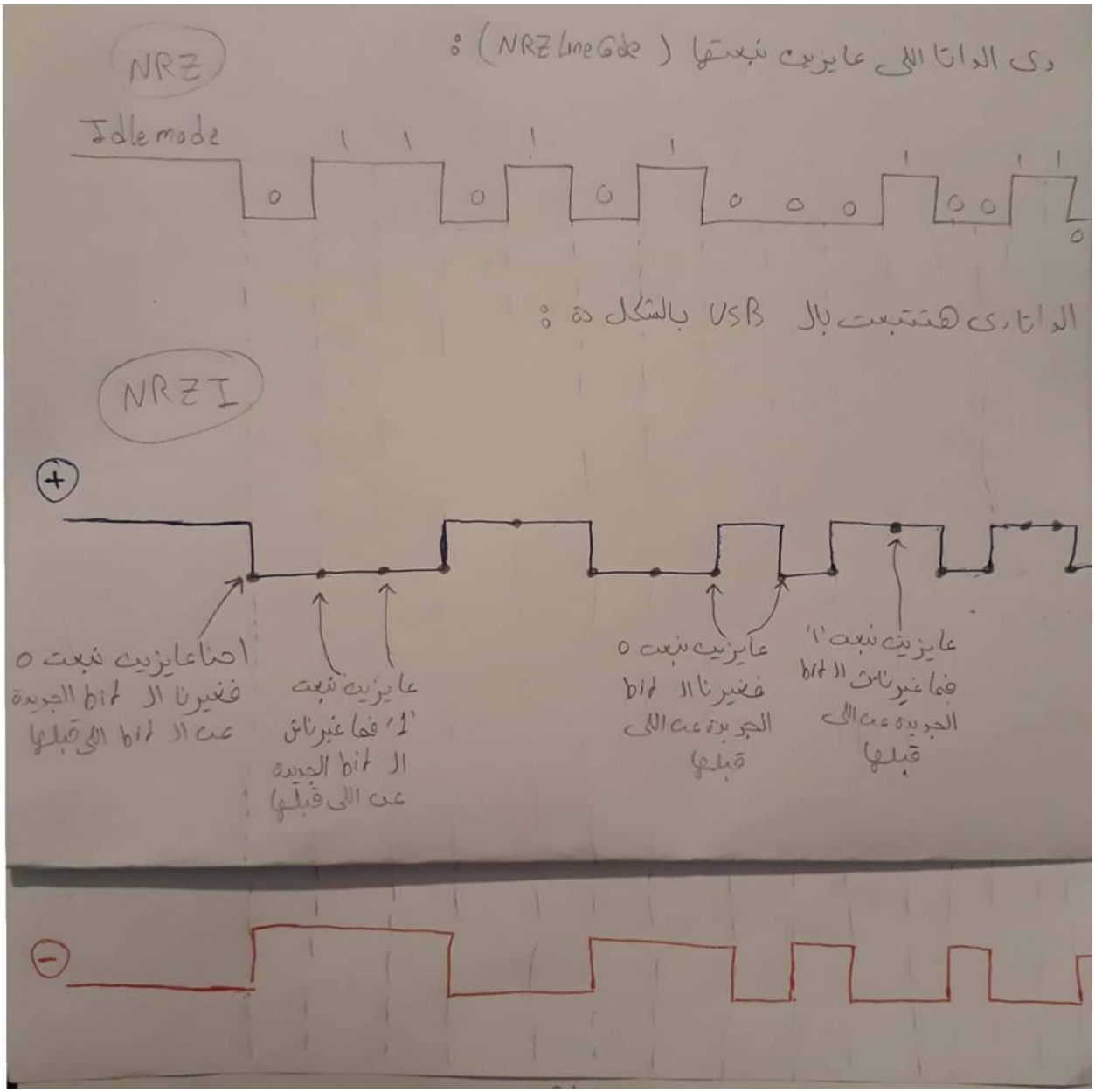
دة طريقة غريبة كدة .. بناخد ال NRZ ونشوف لو ال bit اللي جت لنا دلوقتي هي نفس ال bit اللي جت لنا قبلها، يبقى ال transmitter بعتلنا logic 1

ولو ال bit الجديدة دي كانت غير اللي جتلنا قبلها (يعني كانت 1 وبقت 0 او كانت 0 وبقت 1) يبقى كدة هنعرف ان ال transmitter بعتلنا logic 0

من الآخر لو انا ال receiver يبقى انا بشوف ال bit الجديدة دي كانت ايه ودلوقتي بقت ايه .. لو مختلفين عن بعض يبقى انا هاخد 0 ولو هم الاتنين نفس ال logic يبقى انا هاخد 1

- Differential encoding (NRZI)
  - Zero is encoded as transition
  - One is encoded as no change

تعالى نشوف المثال اللي في slide 55



ال receiver بقى هياخد ال differential NRZI data دي ويترجمها للداتا الاصلية اللي مبعوتة بال NRZ

عايزين بقى نتكلم عن حاجة كمان اسمها bit stuffing .. افرض احنا بعتنا '1' 20 logic كلهم ورا بعض .. ساعتها ال bus هيبقى كأنه في idle mode وبالتالي ال clock هيحصل لها drifting .. فعشان نتجنب المشكلة دي لازم ال bus يشوف ال logic متغير كل 6 clk cycles .. ففي ال case بتاعتنا دي لازم بعد ما نبعت 6 وحايد ورا بعض لازم ال bit السابعة تبقى ب 0 وتبقى ال bit

السابعة دي هي ال stuffing bit .. هتقولي طب ما احنا كدة بنغير في الداتا، احنا مش عايزين ال bit السابعة تبقى ب 0

هقولك ماتقلقش .. ال receiver هيشوف انه جاله 6 وحايد ورا بعض فعلى طول هيعتبر ان ال bit الجديدة اللي هتيجي (اللي هي السابعة) هتبقى stuffing bit "حاجة مش تبع الداتا" فهيرميها "كأنه ماشفهاش" ويكمل عادي بعدها لحد لما يلاقي نفسه بعت 6 وحايد كمان ورا بعض فال bit اللي بعدهم هيعتبرها stuffed وهيرميها وهكذا لحد لما يخلص خالص .. ففي الآخر ال receiver ما أخدش غير ال '1' logic 20 اللي كنا عايزين نبعتهم

ولو هنبعت '0' logic 6 ورا بعض هنعمل نفس الموضوع وهتبقى ال bit السابعة ب 1 وهكذا

ميزة ال stuffing bit هي انها بتـ refresh ال transmission .. بتقوله انت مش idle انت شغال عادي

نخرج من ال stuffed bit بقى .. احنا قولنا في ال USB بنبعت في الأول pattern من ال clock عشان ال receiver يعرف ال clock اللي هيمشي عليها زي ال transmitter .. بس يعني ايه بنبعت pattern من ال clk؟؟ بنبعتها ازاي؟؟! :

تخيل لو احنا بعتنا 1 وبعد كدة 0 وبعد كدة 1 وبعد كدة 0 وبعد كدة 1 وهكذا ... احنا كدة كأننا بنعمل clock ..صح؟؟

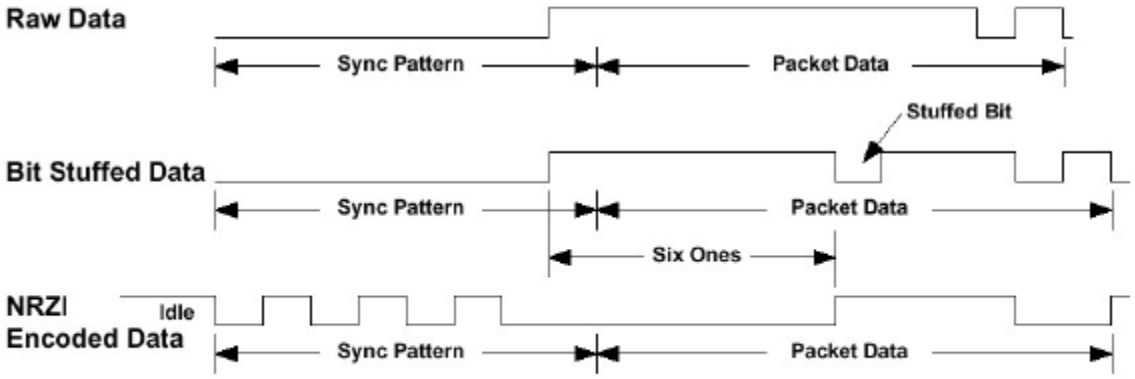
هي دي بقى الفكرة .. ال receiver بقى هياخد ال oscillatory data دي ويشوف ال period بتاع كل cycle جت بكام وبكدة هيقدر يعرف ال bit rate وبالتالي هيعرف ال clock اللي هو محتاج يشتغل بيه

واحنا قلنا اننا لو عايزين ... → 0 → 1 → 0 → 1 يبقى هنبعت بال NRZI ال bits دي : 0000...

فاحنا هنبعت اول 7 bits ب 0 و تبقى ال bit التامنة زي ال start bit اللي هتقول لل receiver استعد هيجيلك داتا

اللي احنا عملناه دة عشان نـ sync. ال receiver مع ال transmitter فهنسمي ال pattern اللي عملناه دة sync pattern .. تعالى بقى نبص على الحوار دة في ال slides عامل ازاي

- **Synchronization field**
  - There is sync field which synchronizes the receiver to transmitter.
  - There are 8-bit sequences
    - 7 0's followed by a 1

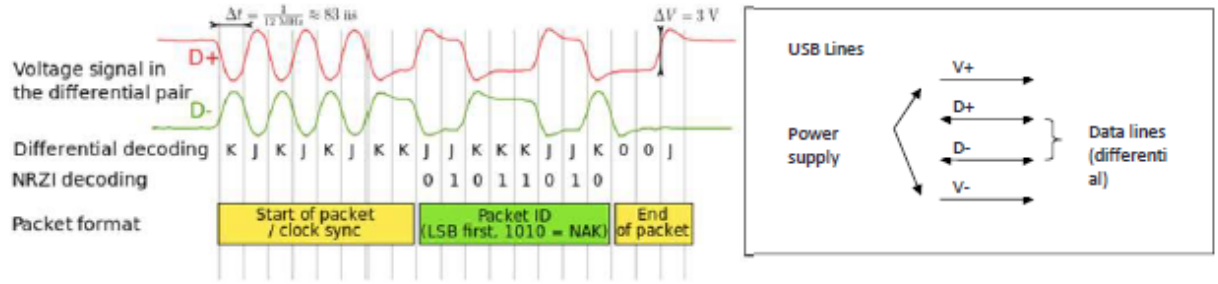


احنا بعتنا داتا (اللي هي raw data) .. الي packet data كان فيها وحايد كتير .. عشان كدة فيه stuffed bit اتحطت بعد اول 6 وحايد .. فالداتا اتشفتت بمقدار 1 (يعني ال 0 اللي كان في ال raw data جه متأخر سيكا)

بعد كدة بقى لما نعمل encoding لل NRZI ... ال sync pattern كان كله اصفار فهيتترجم على انه oscillatory data فال clock ظهرت .. وبعد كدة ال start bit اتحطت .. بعد كدة دخلنا على ال packet data لقيناها كلها وحايد، يعني مش بتتغير .. لحد ما وصلنا لل stuffed bit اللي كانت ب 0 فال packet data اتقلب

بص كدة في ال slide دي .. هو راسم ال differential signaling اللي احنا قلنا عليه .. بس ركز على ال end of packet

- Differential Pair Signaling
  - Differential signaling very good at rejecting common-mode noise. If noise is coupled into a cable, then usually it is coupled into all wires in the cable. This 'common-mode' noise (Vcm) can be rejected by input amplifier.



هتلاقي ان ال end of packet دة single ended مش differential زي الباقي .. ودة لهدف معين اللي هو عشان يقول ان كدة خلاص ال frame كله خلص

نتكلم بقى عن ال data formatting بتاع الأستاذ USB (ال slide دي وقعت مني فلو حد عارف حاجة عنها يقول)

-طبعاً الداتا بتتبعت على هيئة packets

- ال packet نفسها بقى بيبقى فيها Start Of Packet sync. Pattern (SOP) ودة اللي احنا قلنا انه 7 zeros وبعدهم 1

- Packet ID (PID) .. ودة بيحدد نوع ال packet دة ايه (يعني دي data packet ولا نوع تاني) .. دة المفروض انه 4 bits بس عشان يضمن انه بيبعتهم صح فبيبعت ال 4 bits وال complement بتاعهم

- Address field .. دة زي ال addressing بتاع ال AHB اللي خدناه قبل كدة .. ففيه 7 bits بيعملوا addressing لل required device (لأنهم 128 device) و 4 bits بيعملوا addressing لل register اللي جوة ال device نفسه

- frame number field .. دة انا مش فاكر ان الدكتور قال أي حاجة عنه فلو حد فاكر حاجة يقول

- data payload .. دي الداتا اللي احنا عايزين نبعتها وعاملين عليها الفيلم دة كله ... ودي ممكن نحط فيها لحد 1 MB

- CRC : دة نوع تاني من ال error detection .. بياخد ال address field وال data field ويدخلهم على mathematical function بتطلع رقم (بناءًا على ال function) .. لو الرقم اللي اتبعت من ال transmitter هو نفس الرقم اللي ال receiver استقبله يبقى الداتا اتبعتت مظبوطة .. لو الرقم مختلف يبقى حصل حاجة غلط في ال transmission

- End Of Packet (EOP) : دة اللي احنا قلنا انه بيبقى single ended 0 .. لو ال USB شغال في ال high speed يبقى ال EOP بتفضل مبعوتة لمدة 160-175 ns

ولو شغال في ال low speed يبقى ال EOP بتفضل مبعوتة لمدة 1.25 – 1.75 us ... في السلايدز مكتوبة high speed بس دي غلطة في السلايد

لازم أي packet يكون فيه ال SOP وال EOP وال PID .. أي field تاني مش شرط يكون موجود في ال packet (((اعتقد ال address field كمان لازم يكون موجود بس هو مش مكتوب في ال slide ومش فاكر الدكتور قال انه موجود في المحاضرة ولا لأ .. فلو حد فاكر يقول)))

لو ال packet مافيهاش data field هيبقى اسمها token packet .. الدكتور ماتكلمش عنها بس هي مكتوبة في السلايد :

Data Formatting

- Data sent in packets
- Packets will have:
  - Start of Packet Sync Pattern (8 bits, 7 zeros + 1 one)
  - Packet ID (PID) – identifies type of packet. 8 bits total, but only 4 unique bits
  - Address field - 11 bits. 7 bits for USB device (so 128 possible USB devices on bus, host is always address 0), 4 bits for internal use by USB device.
  - Frame number field (11 bits) – incremented by host
  - Data Payload (up to 1023 bytes for high-speed connection)
  - CRC bits - 5 bits for address field, and 16 bits for data field
  - EOP strobe – single ended 0 (160ns-175 ns for high speed, 1.25 us to 1.75 us for high speed)
- Not all packets sent over USB bus have all of these fields (always have SOP, EOP and PID). Packet without data field is a token packet.

مبروك ياجماعة المحاضرة خلصت ☺