حابب أقولكم ان يونيت 3 ده احنا هنتكلم فيه عن ال Embedded C .. فعايز اليونيت ده بالذات تبقوا مركزين فيه قوي .. هو الفكرة كلها ان ال Embedded C هو How to use C programming in Embedded Systems .. فاحنا عندنا Capabilities كتير جداً في ال C .. فاحنا ازاي نطلع كل الكلام ده في خدمة الايمبدد .

فيه حاجات مثلاً في ال C احنا مخدنهاش بس هي بتفيد الناس اللي بيكتبوا applications مثلاً .. زي إنك تعمل حاجات بتعمل Graphics على الشاشة ودي ملهاش علاقة بالايمبدد .. فيه إن انت تقدر ت create file وتكتب فيه وتقرأ منه .. نفس الكلام ده بالنسبة للايمبدد مش مستخدم .

فاحنا خدنا في ال C الحاجات ال Fundamental يعني .. ودلوقتي داخلين على ازاي نركز استخدام ال C على الايمبدد .. طب انت متوقع تطلع بايه من ال unit ده ؟

1- أول حاجه انت تكون فاهم ال processing من أول ما تيجي تقول أنا هبدأ أكتب application لحد ما ال application ده ي run على أي ميكروكنترولر في الدنيا .

2- هتتعلم فيه إنك ت create ال startup from scratch .

3- هتتعلم فيه ازاي تكتب linker script لل ARM .

4- هتتعلم برضو ازاي تعمل Make File وتستخدم ال Debugger .

5- هتتعلم ازاي تعمل Analysis لل binary اللي طالع .

6- ازاي تطلع ال symbol table .

7- ازاي تغير ال entry point وال addresses بتاعت ال sections اللي موجودة في ال binary اللي انت هتحطه على البوردة .

8- وهتطلع منه كمان عارف ايه ال Debugger Circuits اللي موجودة في ال physical boards عشان ت burn أو ت debug أي software بناءاً على أي physical board .

فال unit ده تقيل قوي .. بس هتستفادوا منه كتير جداً .. ده أول Lesson الدنيا فيه بسيطة .. ال Lesson 2 هيبقا مهم جداً .. لأن ده هيبقا تقيل بقا بحيث إن احنا هنتعلم حاجات اللي هو ازاي نكتب startup و linker ونعمل Make file ونستخدم ال GDB .. وهنتعلم الكلام ده على كزا مثال يعني .

فالنهاردة بس هنفتح الدنيا وإن شاء الله ميبقاش فيه مشاكل .

typedef command

أول حاجه هناخد ال typedef .. ده فيه كام حاجه كدا في ال C بنستخدمهم جامد في ال Embedded C .. طب ايه هو ال typedef ؟

ده بالنسبالك هو keyword في ال C .. وده بنستخدمه عشان نطلع alias name ل user-defined data type .. لو فاكر أول session خدناها في ال C .. قولنا ال datatypes دول هما عبارة عن 3 أنواع Primitive و Derived و User-defined .

ال Primitive قولنا انها بتتقسم لحاجتين : integer و real .. وقولنا إن ال integer هو signed و unsigned .. وال real اللي هو ال float وال double وال long double .

ال Derived قولنا ان هما ال union وال pointer وال structure وال array .. وليه هي اسمها derived أو complex ؟ .. دي عشان خاطر انت بتجمع كزا حاجه مع بعض فبتحجز مكان في ال memory على حسب ال shape بتاعك .. بمعنى مثلاً ال structure على حسب ال members اللي انت حاططهم فدول هيترجموا ل قد ايه في ال memory ؟ .. فاحنا مش عارفين .. فيبقى دول كدا derived .. لأن انت بتكون shape جديد من حاجات basics في ال C .. يعني حاجات already مستخدمة في ال C .. زي ال structure ال array ال union ال pointer .

ال user-defined data type .. دي ال user هو اللي بيحددها .. بيحددها بناءاً على ايه ؟ بيحددها بناءاً على primitive and derived data types .. يعني ايه الكلام ده ؟ .. يعني في الاخر هو بيستخدم الاتنين دول مع بعض عشان ي create data types جديدة .. يبقا ال user-defined datatype هو مش حاجه مخترعينها .. هو كل الفكرة إن هو استخدم ال preemptive datatypes وال derived datatypes عشان يكون منهم shape جديد .. ال shape الجديد ده بنسميه user-defined .

وهو لما بيكون ال shape الجديد ده .. بعد ما بيكونه بيديله alias name .. ال alias name ده هو اللي أنا بستخدمه بعد كدا عشان أطلع ال object بتاعتي أو أطلع ال variables بتاعتي .. طب ال syntax بتاعه عامل ازاي ؟

; < typedef < primitive or derived data type > < new_name

ال new_name ده اللي هو ال alias بتاعنا .. بيقولك عشان انت تبقا professional .. فيه بعض القواعد تتبعها في كتابة الكود وده عشان تخليه more readable .. ولكن مش مهم .. يعني ممكن تفكس للقواعد دي :

1- يكون ال alias الجديد اللي انت عرفته ده كله capital .. أو أو character منه capital .

2- عشان تعرف اللي بيقرأ الكود إن ده typedef .. بعد ما بتخلص ال name بتاعك بتعمل _t .. عشان أول ما يشوفوها يعرفوا ان دي datatype معمول ليها typedef .

3- لو هتعرف structure فبتحط في الأول S وال union بتحط في الأول U .

4- ال Enum بتحط ليه في الاخر _e .

دي قواعد لكنها مش إلزامية .

} typedef struct

; uint32_t weight

; Sperson_t {

هنا كإنك بتقول اعملي typedef للحتة المتظللة بالأصفر دي .. وخلي ال alias بتاعها Sperson_t .

وبعد كدا بقا هتستخدم ال Sperson_t دي في تعريف types منها بقا هتكون زي ال struct ده بالظبط .

ال typedef بنتسخدمها في ال AUTOSAR .. يعني هو مثلاً عامل specification لل platform types .. يعني انت مثلاً هتيجي تعمل environment هيكون فيها ال BSW (Base Software Layer) ودي اللي بيكون فيها ال COM Stack وال MCAL .. وهتعرّفلي حاجه اسمها TTC .. ودي حاجات ليها علاقة بال compiler وفيه startup وفيه application .. وال application ده اللي هيبقى فيه ال software components بتاعت ال SWC (customer) .. يعني دي كدا ال AUTOSAR Stack .

ال Application Folder بيكون فيه حاجات ال Customer .. ال TTC فيه الحاجات بتاعت ال Compiler .. وال BSW جواه بقا ال OS مع ال COM Stack مع حاجات كتير قوي .. فبيقولك بقا ان من ضمن الحاجات .. فيه حاجه بنحطها جوا ال TTC اسمها ال Platform types .. ودي بنعرف فيها الحاجات ال common .. يعني ال header اللي أنا هتسخدمها في أي حاجه .. ممكن أستخدمها في ال BSW .. ممكن أستخدمها في ال application .. ممكن أستخدمها في ال TTC .

فال platform types دي فيها file .. ال file ده هو اللي هنركز عليه دلوقتي .. ده معمول فيه typedef لحاجات كتير قوي متسخدمة في ال AUTOSAR .. والبشمهندس حاططلك في ال Slides ال Specifications بتاعت ال AUTOSAR دي لو انت عايز تنزلها يعني وتتفرج عليها .

هتلاقي فيها Application Layer فووق خالص .. تحتها RTE (Runtime Environment) .. تحتها BSW (Basic Software Layer) .. تحتها Microcontroller .. فدول كدا ال 3 layers بتوع ال AUTOSAR .

هو بقا بيقول إنك سواء كنت Non-communication أو communication basic software .. الفرق بينهم إن ال communication بيستخدم ال COM Stack لكن ال non-communication مش بيستخدمه .. سيبك من الكلام ده هنفمه في ال AUTOSAR إن شاء الله .

اللي أنا عايزك تركز عليه إن الاتنين Software Modules والاتنين بيعملوا include لل Std_Types.h .. ودي لازم ت include ل compiler.h و Platform_Types.h .

و Platform_Types.h ده الفايل اللي فيه type definitions لحاجات كتير على حسب البوردة .. وده اللي هن focus عليه دلوقتي .. الكلام ده مش البشمهندس اللي مخترعه .. لأ ده أي حد هيتعامل مع ال AUTOSAR هيلاقي إن ال modules بت include ال Std_Types.h .

مثال .. يعني انت لو فكرت مثلاً تعمل MCAL .. هتلاقي في الآخر إن انت ال MCAL بتاعك ده أو ال Driver بتاعك ده بي include حاجات واصلة في الآخر ل Std_Types.h ودي واخدة معاها ال Compiler.h وال Platform_Types.h .

ايه بقا اللي عايزين ناخده من الحتة دي ؟

فيه snapshot في ال slides متاخدة من ال AUTOSAR Specification .. هو عاملك type definition ل 3 architectures مختلفين .. هو كان عامل كزا architecture .. عامل ل ST30 .. و SHx ودي microprocessor نوعها SH يعني Renesas ك company عاملة System on-chip يعني ال processor هي اللي عاملاه واسمه SH .. فالمهم يعني إن ال processor بتاعها من architecture اسمه SH .

وفيه برضو architecture اسمه ST .. وفيه architecture غيرهم برضو تبع Freescale اسمه S12 .. وفيه برضو حاجات ARM وحاجات كتير زي أنواع ال processors اللي انتوا عارفينها .

ال AUTOSAR قاليك في ال Specification بتاعه في ST30 لما تيجي تعرفلي boolean مثلاً تعمله كدا :

```
typedef unsigned long boolean ;
```

طب ايه ده ؟ .. مش المفروض ال boolean ده بيكون 0 و 1 ؟ .. انت معرفهولي على إنه unsigned long ليه ؟

هو ال AUTOSAR اللي قايل كدا .. انت كدا كدا في الآخر اللي انت هتستخدمه هو bit واحدة منه .

عندك في ال SHx بقا هو معرفها كدا :

```
typedef unsigned int bool ;
```

وعند في ال S12 هو معرفها كدا :

```
typedef unsigned char boolean ;
```

هتقولي يا بشمهندس طب ليه ما يعملهمش كلهم unsigned char ويريح دماغه .. خلي بالك إن الكلام ده كمان ليه علاقة بال assembly .. يعني ايه ؟

هو اللي كاتب ال Specifications راح مثلاً شاف ال Compilers بتطلع assembly code مكتوبة ازاي .. لو فاكر في ال structure كان فيه حاجه اسمها store-word و store-halfword .. احنا خدنا إن stw مثلاً كانت بتكتب 4 byte .. كانت أسرع من stb .. يعني فيه موضوع alignment .. وده معناه إن ممكن إن ال processor بالنسباله عنده instruction واحدة تكتب على 4 byte أسهل من إن هو يقعد يكتب على byte byte individually .

واحنا خدنا ليها مثال في lecture ال structure .. لما احنا شيلنا ال packing وخليناها padding .. وبعد كدا حطيناها packing .. لقينا ايه اللي حصل ؟

لقينا إن الحاجه اللي كانت بتتعمل على instruction واحدة .. اتعملت تقريباً على 3 instruction معايا .. فده معناه إن ك execution time بقا أكبر .. فبتوع ال AUTOSAR بيدرسوا إن بالنسبة لل Processor هل أسهل ليه في ال Execution إن هو يستخدم instructions تروح لل word كلها مع بعض ؟ ولا إن هو يروح ل individual bytes .

اللي أنا عايز أقوله إن ال typedef مش حاجه واحدة .. يعني اوعى تاخدلي typedef تحطهولي على جميع أنواع ال architectures .. لأ هو الفكرة هو انت بتعمل ال typedef بتاعك ده Compatible مع ايه ؟

كل architecture ليه ال alignment بتاعته وليه ال Pulse Width بتاعه وليه كزا حاجه .

سؤال كدا مع نفسي ك عبدالله .. هو احنا بنعرف بقا ال alignment والحاجات دي منين بتاعت كل architecture ؟

بجاوب على نفسي .. تقريباً من ال Specifications .

احنا فهمنا دلوقتي إن ال typedef ده مهم .. وإن احنا بنعرفه بإيدينا وحتى ال AUTOSAR بي recommend ازاي تعمل typedef لأسامي هو بيستخدمها وهو قايل عليها .. وعرفنا كمان ليه في architecture معين استخدم long وفي واحد تاني استخدم character .

نلاحظ ايه تاني في الموضوع ده ؟

لما هو جه يستخدم ال sint32 قالك :

```
; typedef signed int sint32
```

وفي architecture تاني :

```
; typedef signed long sint32
```

و في architecture تالت :

```
; typedef signed long sint32
```

يبقى دلوقتي ال int في واحدة منهم ممكن تبقا 4 byte وفي واحدة تانية تبقا 2 byte .

طب الأسامي بتاعت ال alias دي لازم أسميها زي ما هي كدا ؟

لو انت الكود بتاعك aligned مع ال AUTOSAR بيقا لازم تلتزم بالأسامي دي .. يعني لازم وانت بتعمل ال typedef بتاعك تلتزم بيهم .. لازم يكونوا عندك بالظبط زي ما هم معمولين في ال Specification .

الفكرة كلها إن أنا عايزك تعمل الأسامي دي بإيدك وتخليها عندك بحيث إن انت لما تيجي أي بوردة من نفس ال architecture تستخدمها .. إنما لو ال architecture اتغير .. ساعتها خش على ال specification وشوف ايه الأحسن لل type definition ك types يعني تستخدمها .

مطلوب منك تعمل الحاجات دي .. أول assignment معانا في ال Lecture دي هي إنك ت Create file اسمه Platform_Types.h زي اللي البشمهندس عامله .. (أنا عملته الحمد لله) .. وتحطه على ال Repository عشان ده بعد كدا هنعمل include ليه في أي بروجكت هنعمله ونستخدمه .

فيه حاجه اسمها ال " stdint.h " .. ده header موجود في library جوا ال toolchain أو ال compiler اللي انت بتستخدمه .. وده فيه كل تعريفات ال type definitions للحاجات المستخدمة أو ال famous .

انت لما تيجي تعمل ال Platform_Types.h ممكن تستخدم ال header ده أو لأ يعني براحتك .

ده كدا احنا اتكلمنا عن ال typedef .

Header Protection

ايه ال header protection ده ؟ .. لو احنا بصينا مثلاً على المنظر ده .. لو أنا جيت مثلاً في الكود بتاعي عرّفت file اسمه t.h و a.h و b.h .. وجيت مثلاً في ال t.h قولت أنا عايز أعرّف typedef ل struct .. وتعالى نقول إن ال struct ده فيه variable من نوع int واسمه x .. وبعد كدا عرّف الكلام ده على إنه S_C .

```
typedef struct { int x ; } S_C ;
```

هاجي بعد كدا وأروح لل a.h وكل اللي هعمله في ال file ده إني هعمل include لل "t.h " وهنعرّف فيه function مثلاً اسمها af ودي بتاخد ال data types اللي اسمها S_C

```
void af ( S_C q ) ;
```

هروح لل t.h وأعمل نفس الكلام بس هخلي اسم ال function ل bf وهي بتاخد برضو S_C

```
void bf ( S_C q ) ;
```

بعد كدا لما نيجي نروح لل main وهنعمل كدا :

```
#include " stdio.h " #include " a.h " #include " b.h"
```

; int main ( ) { S_C r ; af (r) ; bf (r) ; return 0

{ ; (“ void af (S_C p) { printf (“ af

{ ; (“ void bf (S_C p) { printf (“ bf

فالفكرة بقا لما أجي أعمل build للكلام ده .. هيجيبلي error وهو إن S_C معمول ليها previous declaration .. يعني فيه duplicate structure definition لل S_C .. طب هو ليه بيحصل الكلام ده ؟

مش احنا قولنا في عملية ال preprocessing لما احنا بينّا ال i. فايل لما طلع .. لقيناه إن هو شال ال #include وحط مكانها ال text replacement بتاعته .. وقولنا إن ال i. فايل ده هو اللي بيخش بعد كدا على ال compiler .. فده معناه ايه ؟ .. ده معناه إن ال preprocessing في ال compiler هيشيل ال #include الأولى ويحط اللي جواها .

فلما ييجي يحط اللي جواها فهيتحط t.h جوا ال a.h وبرضو مع ال b.h .. ففي ال main أنا عامل include للاتنين والاتنين فيهم ال t.h اللي فيها ال declaration .. فال compiler لما خد الكلام ده عشان يعمله declaration .. ايه اللي هيحصل ؟ .. هيلاقي إن ال typedef دي اتكررت مرتين في ال c. فايل .. فيديك compilation error ويقولك إن ده previous declaration .

طب حاجه زي كدا هنتغلب عليها ازاي ؟

فيه مشاريع مثلاً زي ال Linux Kernel فيه ناس بتكتب فيه درايفرات .. ده بيوصل مثلاً في range (لو انت بتتكلم على بوردة واحدة) .. من 250 لحد 400 فايل .. وده احنا بنتكلم على بوردة واحدة .. أما هو أصلاً ال Linux Tree اللي انت بتنزلها لم حسبوها فهي 25 مليون سطر C .. كل دي فايلات من غير أي Compilation .

فاللي أنا بقوله .. انت ممكن تبقا في مشروع بتكتب درايفر أو بتكتب Component أو Application .. مش هتقعد تشوف بقا مين بيعمل include لمين عشان توصل لإن ممكن مشكلة زي دي تحصل .. امال انت بتعمل ايه ؟ .. بتعمل حاجه اسمها header protection .

ودي نظامها كالآتي .. انت ملكش دعوة مين بي include مين .. أول لما أجي أكتب أي header file هعمل كدا مثلاً لل a.h :

#ifndef A_H_ #define A_H

#endif

وهكذا مع باقي ال header اللي في البروجكت بتاعك .. طب انت كدا عملت ايه ؟ .. انت كدا حميت نفسك .

يبقا انت في أي h. فايل تريح دماغك وتعمل protection لل header بتاعك .. وكدا لو جينا عملنا build تاني فهو هيتم عادي .

خلي بالك إن الحتة اللي بيكون معمول عليها gray في ال STM32CubeIDE أو Eclipse .. فده معناه إنه في ال Pre-Compile هيشيلها .. فبقا كل الناس اللي بتعمل drivers بتعمل ال header protection بالمنظر اللي احنا قولنا عليه ده .

Optimization

ال optimization ده by default إن ال compiler عامل ال optimization level بتاعه O0 .. هو انت عندك level 3 من ال optimization وفيه بعض ال compilers بيزودوا level زيادة .. ال optimization level دي هي في الآخر بتتباصى لل compiler اللي هو gcc على إنها :

- O

وبعد ال O دي بقا بتحط 0 أو 1 أو 2 أو 3 .

ال Optimization ده معمول عشان خاطر ايه ؟ .. معمول عشان خاطر ي reduce ل 3 حاجات :

1- Number of instructions (text section)

ال Number of instructions ده يعني ال txt. سكشن اللي طالع .. ال binary اللي بيطلع بيتقسم ل sections .. هنعرف الكلام ده بالتفصيل .. ال txt. سكشن ده فيه كل ال instructions أو ال assembly بتاعت الكود بتاعك .. فال txt. ده لما بقلله بيقا أنا كدا قللت ال code size ك overall بقا اللي هو ال binary النهائي .. فال binary النهائي بياخد مساحة أقل في ال Flash memory .. ومش بس كدا .. ده أنا لما بقلل ال instructions فأنا كدا بخلي ال processor ي execute أسرع .

2- Memory Access Time

فيه بعض ال optimization algorithms اللي بيعملوها في ال compiler .. يلاقيك مثلاً انت بتعرّف variable وبعد كدا انت استخدمت ال variable ده انك تدخله في equation وبعد ال equation خدت الناتج ضربته في variable تاني وبعد كدا سجلته في ال memory .

مثال .. أنا عندي مثلاً x .. ال x ده انت قريته من ال memory .. وبعد ما قريته من ال memory ضربته في equation كبيرة .. بعد كدا الناتج اللي طلع استخدمته في equation تانية .. بعد كدا الناتج اللي طلع استخدمته في equation تالتة .. بعد كدا الناتج اللي طلع حطيته في ال memory .

خلي بالك إن الحتة دي بتختلف من compiler للتاني .. يعني فيه شركات وظيفتها وأكل عيشها بس إن هي بتعمل compiler .. ففيه ناس تيجي تقولك أنا عندي optimization algorithm شغالين عليه بتيم بيتكون من 30 Engineer يقدر يقلل ال memory ل percentage كزا .. طب ليه هو عمل كل الهيصة دي ؟

أصل انت لو هتكتب انت assembly بإيدك أنا كنت هعمل كدا .. هروح أقرا من ال memory ي Load .. وبعد كدا هاخد ال value اللي طلعت من ال equation دي .. الناتج هروح أسجله في ال memory .. وبعد كدا الناتج اللي هسجله في ال

memory ده هستخدمه تاني وأعمله Load عشان أضربه في equation تانية عشان الناتج اللي طالع ده كمان هسيفه في ال memory وبعد كدا هاخده أعمله Load من ال memory وأضربه في equation تالتة .. وبعد كدا هعمله save في ال memory النهائية .

فهو يقولك بقا طب ليه الهيصة دي كلها ؟

ممكن ييجي ال compiler عند ال optimization يعمل كالآتي .. الناتج اللي طلع أول مرة راح نزله في Accumulator register أو في general purpose register مثلاً R16 .. وبعد ما عمل ال equation وطلع ناتج تاني راح خزنه في general purpose register غيره .. بعد كدا الناتج خده حطه في general purpose register برضو وبعد كدا حطه في ال memory .

لو احنا بصينا كدا .. هنقول هو وفر ايه ؟ .. خلي بالك ال general purpose registers دي موجودة جوا ال CPU نفسه .. وال CPU نفسه لما كان بيروح لل memory كان بيروح على Bus مثلاً AHB .. بعد كدا راح لل memory اللي بيعمل save فيها .. فبدل ما يروح الطريق ده كزا مرة .. هو لما عمل save في registers جوا ال processor نفسه .. فهو كدا وفّر interaction مع ال memory اللي برا كزا مرة .. فدي تخلي ال performance ك overall أحسن .. وال power consumption أقل .

طب هو دي فيها صعوبة يا بشمهندس ؟

اه طبعاً .. دي بالنسبة للبشمهندسين اللي بيكتبوا ال compiler فيها صعوبة .. المهندسين دول بيبقوا مذاكرين الكود كويس قوي .. وعاملين algorithm إن هو مثلاً ال core ده بيتكون من 32 general purpose registers .. خلي بالك هما عاملين algorithm إن هو بي translate الداتا .. فهو بيستخدم ال general purpose دول .. فهو ال algorithm بتاعه المفروض يحافظ إنه لو استخدمهم كلهم ما يستخدمهمش تاني .. أو يكون محافظ إن هو يكون موفر لنفسه ال general purpose كل شوية عمال يفضيه عشان على مدار الكود يستخدمه في كزا حاجه .. فده ال algorithm بيبقا فيه جزء processor architecture وبيبقا فيه جزء direct memory وبيبقا فيه جزء assembly .. يعني جزء صعب جداً .

لكن احنا بالنسبة لينا كمهندسين بنكتب bare-metal أو درايفرات أو C فالجزء ده مش يهمناش قوي .. مثلاً بيبقا فيه ناس شغالة ك ARM .. يعني ايه ؟ .. يعني هو بيروح يكتب assembly على ال ARM دي كل وظيفته .. يا إما startup .. يا إما بيعمل حاجات ليها علاقة بال compiler .. دي بتبقا شغلته .

احنا كمهندسين Embedded بقا ال Scope بتاعنا اخرنا ال C .. أي حاجه C هتترجم بقا .. دي بتكون بتاعت ال Compiler مش بتاعتي .. أنا كمهندس بكتب Embedded في شركة مش هتفرق معايا الحقيقة هو هيستخدم Direct memory access ولا indirect .. أنا مش فارقه معايا .. دي حاجه بس هتفرق لو أنا ال software بتاعي هو حساس جداً لمستوى إن أنا هتعامل مع كل cycle بحذر .. بس قليل قوي لما تلاقي كدا .. يعني بتلاقيها في applications بيبقا ليها علاقة بالحاجات ال medical قوي أو صواريخ والكلام ده .. لكن في ال Automotive والعربيات مفيش الكلام ده .

Power Consumption -3

انت لما بتعمل أول خطوتين اللي هما تقلل ال instructions وتقلل ال memory access فانت كدا بتقلل ال power .

طب ازاي نتحكم في ال optimization بتاع ال compiler ؟

بص هو Eclipse ده Open-Source وكمان ممكن تاخد أي tool chain تنزلها وتعملها import جوا Eclipse وتشتغل بيه على أي نوع architecture .. ومش بس كدا .. Eclipse كمان عشان هو Open-Source .. فيه شركات خدته وبنت عليه حبة Plugins بال java عشان في الآخر يعملوا product جديد وبيبيعوه .. يعني شركات بتاخد Eclipse ده بتبدأ تزود عليه Plugins وفي الآخر تقفله وتطلعه ك product باسم تاني .. فيه حاجات ليها علاقة بال AUTOSAR وحاجات ليها علاقة بال ARM وحاجات ليها علاقة ببوردات معينة .

طب احنا دلوقتي عايزين نغير ال optimization level نعمل ايه في الآخر ؟

هنيجي على ال project بتاعنا في Eclipse وندوس Right click ونختار properties وبعدها نختار C/C++ build ثم settings ثم tool settings .. وجوا ال tool settings دي هتلاقي بقا ال GCC C Compiler .. هتلاقي فيه جزء ال Optimization .. هيجيبلك ال Optimization level .

1. -Null (O0)
2. -1 level (O1)

وهكذا

طبعاً الكلام ده بيترجم في ال command اللي بيديه ل gcc .. يعني انت لو دوست على GCC C Compiler هتلاقي :

Command : gcc All options : -O0

فيه options كمان بس أنا مكتبتهاش .

فهو كدا بيقوله يا gcc compiler خد option بتاع -O0 .. ولو أنا جيت في ال optimization وقولت ليه -O1 وعملت apply .. لو روحت لل GCC C compiler هتلاقيه عملها -O1 .

يعني أنا لو معايا terminal وبكتب أنا بإيدي gcc أقدر أقوله -O0 أو -O1 أو اللي أنا عايزه براحتي .

Optimization level O0

ال level ده مش بيعمل حاجه .. وده مش recommended ان انت تعمل بيه production لو انت كان عندك limited memory .. انت لو معندكش limitations ممكن تستخدمه عادي .

ده بيعمل fastest compilation time .. خلي بالك يا صاحبي .. فيه فرق بين حاجتين .. ال execution time وال compilation time .

ال compilation time هو ال time اللي خده عشان ال compile يوصل إنه يطلعلك executable file .. اللي هو ال binary النهائي .

ال execution time ده بيكون أنا خدت ال binary النهائي حطيته على البوردة وبعد كدا شغلت البوردة .. فبنشوف هي خدت execution time أو running time قد ايه .

فهو بيقولك طبعاً ما دام مفيش optimization يبقى هو fastest compilation time .

ال level ده برضو بيبقا Debugging friendly وخلي بالك من الحتة دي كويس قوي .. ليه بقا ؟

بيجي مهندس كان كاتب حاجات كتير قوي في الكود .. وبيجي بعد كدا يطلع كود فالكود ما يشتغلش .. يبدأ ي debug الكود .. وهو بي debug الكود يقول والله أنا كنت عامل symbol اسمه x .. أنا مش لاقي ال x ده .. كنت عايز أشوف قيمته بتتغير بقد كام .. فيلاقي إن هو كان ال optimization مفتوح .. فلما كان مفتوح لقى إن ال x ده مش بيستخدم كتير فشاله .

فبيقولك بقا خلي بالك لو انت بت debug ياريت ترجع الكود بتاعك ل optimization level O0 عشان ما يكونش شالك حاجه من الحاجات المهمة أو حاجه من الحاجات اللي انت أصلاً بتعملها monitor .. على فكرة ساعات ال optimization بيشيلك حاجه مهمة بالنسبة ليك في الكود .. يخليلك ال logic بتاع ال software بتاعك كله يبقا غلط .

ساعتها بقا ما تقوليش أنا ه debug وأنا قافل ال optimization .. مانت هتلاقيه صح ساعتها .. لما تيجي ت debug وانت قافل ال optimization مش هيمسح الجزء اللي عملك الغلط .. فهتلاقي الكود شغال صح .. فتقعد بقا تضرب كف على كف .

فانت بقا هتقارن ال assembly code ما بين ال 2 versions .. ال version اللي فيها optimization وال version اللي مفيهاش optimization .. وتحطلي الاتنين على موقع اسمه Dev online .. وحط فيه ال 2 assembly files بتوعك وشوف أنهي section اتمسح .. وهيكون ال section اللي اتمسح ده هو اللي عملك المشكلة دي .. وساعتها تخش تعمله volatile وتقفل الدنيا .. فخلي بالك من حاجات زي كدا .

Optimization level O+

ال O+ دي يعني O1 , O2, O3 وهكذا .

Optimization level O1

ده بي moderate ال optimization عشان ي decrease ال memory access .. يبقى ده بس على مستوى ال interaction مع ال memory .

Optimization level O2

ده بيكون Full optimization و Slow compilation time .. بس هو Not debugging friendly .. وده معناه إنك لما تيجي ت debug هتلاقي الحاجات اللي انت كنت عايز تشوفها ملهاش symbols .. فهي بالنسبالك assembly pure كدا .. وانت مش فاهم ده بتاع ايه .

Optimization level O3

ده بقا كل حاجه في الدنيا .. وهو عباة عن ال O2 + بعض ال aggressive steps .. طبعاً الكلام ده بيختلف من compiler للتاني .. الكلام ده مش ثابت .. فيه compilers بتعمل ال aggressive steps .. يعني بتمسكلك الكود حتة حتة .. لو فيه فتفوته مش مستخدمة أو استخدامها ضعيف بالنسبالهم هتمسحهولك .. وده بيكون ال slowest execution time .. وساعات ممكن يعملك bugs في ال program بتاعك وبرضو بيكون Not debugging friendly .

لو الكود بتاعك انت كنت مديله Optimization level 3 وبعد كدا لقيته مش شغال .. فأول نصيحة شيل ال Optimization level خالص .. خليه no optimization وتشغله .. لو اشتغل يبقا خلاص .. لكن لو ما اشتغلش يبقا هو فيه مشكلة كبيرة هنقعد نسهر بالأيام عشان نكتشفها .. أما لو هي بس مشكلة optimization فاحنا بنحمد ربنا وبنبقا فرحانين جداً .. وساعتها بنقارن ال 2 assembly ببعض .. نقارنهم ونطلع الفرق ونبدأ نصلحه .

Keil Micro-vision

من ضمن ال assignments اللي علينا في ال Lesson ده هي تسطيب ال tools .. وأول tool معانا هي Keil Microvision .. احنا النسخة اللي منزلينها Non-commercial .. يعني ال binary اللي طالع ده مش هحطه على product .. فهو عشان كدا Non-commercial .

ال Keil ده رغم إن ال Editor بتاعه مش لطيف .. بس فيه ميزة جميلة .. إن هو بتاع ARM .. يعني ARM هي اللي عاملاه .. فدي نقطة .. تاني نقطة .. إن هو بي provide ليك simulation .. يعني انت في ال debug تقدر ان انت ت use simulator .. فتقدر ت simulate الكود بتاعك من غير ما يبقى عندك physical board .

هو نزوله سهل وتسطيبه سهل .. اللي هيتعبك إن فيه حاجه جواه اسمها ال Package installer .. هتاخد وقت كبير عشان تنزل الحاجات اللي انت عايزها منه .. يعني البشمهندس بيقولك حدد ال packages اللي ينزلها وسيبه ينزل وخش نام .

طب انت ايه ال packages اللي هتنزلها ؟

ممكن نقول ال startup وال linker وال driver اللي يخليك تعرف ت burn الكلام ده على ال physical board .. ومش بس كدا .. فيه جزء كمان ليه علاقة بال simulator .. فيه فيديو نازل هتلاقي فيه تسطيب كل حاجه انت محتاجها .

المهم انت هتعمل search على ال tiva c .. وهتدوس عليها .. وهيظهرلك على اليمين ال packages اللي المفروض تتحمل .. سيبك من ال generic ما تبص عليها .. ال generic ده يعني كل حاجه في الدنيا .

انت هتبص على Device Specific .. وتعمل install للي هيظهرلك .. ولو انت محملهم قبل كدا هيظهرلك up to date .

يبقى دا كدا ال tiva c .. تاني حاجه بقا ال STM32 .. احنا هنشتغل Cortex-M4/M3 .. فانت هتكتب STM32F4 وتختار كل ال Packages بتاعتها في ال Device Specific تعملهم install .. وبرضو هتعمل STM32F1 وتعملهم install .

بعد ما تعمل ده كله .. هتيجي تعمل new uvision project وتختار المكان اللي هتحط فيه ال project بتاعك .. هيقولك انت عايز تشتغل ببوردة ايه .. هتقوله STM Microcontrollers ثم STM32F1 ثم STM32F103C6 وهي دي اللي هنشتغل عليها مثلاً .

هيديك شوية options .. مثلاً في ال Device هتلاقي فيه إنك تقدر تختار إنه يضيفلك startup ولا لأ .. لو علمت صح عليها يبقا انت كدا مش هتكتب startup وهو هيطلعهولك .. بس احنا في حالتنا مش عايزينه .

هيفتحلك project بيكون فاضي مفيهوش أي حاجه .. انت هتدوس بقا كليك يمين وتقوله New item أو existing item .. وتبدأ بقا تحط ال startup وال linker وال main file بتاعك وكل حاجه .. لأن أنا خدته فاضي خالص .

لما تيجي ت debug بعد كدا بتقوله .. Use simulator في ال Debug .. وكدا لما تيجي ت compile و ت debug الدنيا هتمشي معاك .

نصيحة .. لو هتشتغل على Keil Uvision لو انت هتستخدمه مع STM32 لأن هو ال Editor بتاعه بشع .. (تقريباً قدام البشمهندس هيقولنا نكتب الكود على STM32CubeIDE وناخد ال elf. ون debug هنا) .

STM32CubeIDE

دي برضو محتاجين ننزلها .. خلي بالك دي معمولة لل STM32 مخصوص .. وده بيكون based على Eclipse .. فهما يعتبر واخدين Eclipse ومزودين عليه Plugins وطلعوا tool اسمها STM32CubeIDE .. وال tool دي Free و Open-Source .

أنا هستخدم ال tool دي عشان أ compile وأكتب الكود .. وال binary اللي هيطلع أو ال executable file اللي هيطلع ده .. هروح أشغله على ال Uvision عشان أشغله على ال simulation وأشوف ال processor registers بتاعته عاملة ازاي .. ويبقى أنا كدا استخدمت ده عشان أ compile حاجتي .. واستخدمت التاني عشان أ debug .

لكن لو معاك ال Physical board فانت مش محتاج Keil في حاجه .. ممكن تحرق علطول عليها من STM32CubeIDE .

طب يا بشمهندس هو ينفع من غير أي حاجه أ compile ال ARM وأحطه على البوردة ؟

اه ينفع عادي .. ال IDEs دي كلها أصلاً هي عبارة عن Editors وحبة حركات كدا + إن فيه toolchain أو compiler موجود behind ال IDE ده .. يعني شغال في ال Background .. هو في الآخر انت بتروح تندهه عشان ي compile حاجتك .

فاحنا برضو في الكورس ده بتاع ال Embedded C .. هنتعلم ازاي من غير ما نستخدم اي IDE في الدنيا إننا نكتب كود ونكتب startup و linker وبإيدينا ن compile من ال terminal ونطلع executable نروح نحرقه .

عشان انت ممكن مثلاً تبقا في شركة بتتعامل مع بوردة معينة .. والبوردة المعينة دي ملهاش tools نازلة أو ال tools اللي نازلة دي بفلوس كتير قوي .. فالشركة تقولك انت مش مهندس يا حبيبي ؟ .. احنا ليه نشتريلك tool عشان تكتب عليها ؟ .. ما تجيب ال toolchain من على النت وتكتب انت الكود وت compile وتطلعلي الفايل ده .. انت ليه عايز IDE ودلع ؟

فساعتها هتقولهم خلاص .. أنا مهندس Embedded وهعرف أشغل الكلام ده من غير IDE .. فده اللي هنتعلمه برضو في ال project الجاي .

احنا هنتعلم نكتب ال startup و ال linker .. لكن لو انت في أي شركة مش هتكتبهم .. هو بيكون فيه ناس تانية مخصوصة إن هي تكتبهم .. الناس اللي صنعوا البوردة أو الناس اللي عاملين ال toolchain أو بعض الناس ال developers اللي بيعملوا optimization لل boards .

ولكن انت كمهندس Embedded بتبان شطارتك ان انت تعرف تقراه .. أو إن لو حصل إن انت محتاج تعدل فيه حاجه بإيدك تقدر تعدلها .. تدوس معاه .. مش تقوله لأ دا الحتة دي معرفهاش .. مهندس ال Embedded ما يقولش ما أعرفش حاجه .. يخش بإيده ويعدل ويعمل .. بس حالياً ال startup وال linker بيكونوا جايين ك support package للشركات .. ولكن ممكن انت تشتغل في الشركة اللي بتعمل ال support package بقا .

أول ما بتفتح ال IDE بتاع STM32 هتلاقي فيه حاجه معمولها ماكرو على ال Soft Floating Point وال ARM Floating Point .. دي هناخدها لما نيجي ناخد ARM Assembly .. مش هينفع نقولها دلوقتي .

هتلاقي في ال slides برضو فيه مثال على الفرق ما بين ال sizes في ال text لما نعمل build .. هتلاقي في ال optimization مسح الكود ال redundant وال size بقا أقل .. وال hex اللي بيطلع برضو قل .

أنا كمان لما بصيت على ال binary اللي طلع وشغلته على ال simulation بتاع ARM جابلي كمان عند كل سطر ال assembly بتاعه عامل ايه .. يعني هو مسح data2 و data1 .. لأنه دخل جوا اللوب لقى ملهومش أي لازمه .. واحنا عرفنا الموضوع ده عن طريق ال debug .

طب نعمل debug ازاي ؟

ال B في ال assembly يعني Branch .. فهو في المثال اللي احنا بنتكلم فيه هيعمل branch على ال address اللي هو 0x080001ES واللي هو ال main في حالتي هنا .. فهو عمال يعمل branch على نفسه لأن هو مسح الكود اللي فوق اللوب وساب اللوب بس اللي هي while (1) .. فهيقعد يعمل branch على نفسه في infinite loop .

الحاجات دي لا تكتشف إلا بال debugging .

خلي بالك تقريباً السطور بتكون مترقمة في ال debugging زي ما هي في الكود الأصلي .

لو احنا بقا عملنا ال data2 , data1 من نوع volatile .. فهو هينفذ ال swap غصب عنه .. وبعد كدا هيدخل في ال infinite loop .. من هنا بقا جات فكرة إن أنا قولتلك نزل ال Keil Uvision .. ليه ؟ .. عشان نعمل الحركة دي .

نكتب الكود ب IDE سهل بالنسبالنا و powerful و ال syntax فيه كويس .. وبيخلي فيه Capabilities إني أظبط الكود بال CTRL + H وال CTRL + I وال CTRL + R والكلام ده كله .. ونطلع ال executable نعلمه debugging على ال Simulator بتاع Keil Uvision .

طب ازاي تعمل حاجه زي كدا ؟

هنيجي ناخد ال executable كوبي من ال STM32 .. ونروح في ال Keil Uvision .. ندوس على Project ثم Options for target .. فيه من ضمن ال options هي ال Output .. وده معناه انت بتطلع ال binary بتاعك فين .. هتاخد ال executable وتعمله paste وهتخلي ال extension بتاعه axf. بدل من elf. ويبقا انت كدا تقدر تعمل debug على الكايل يا برو .. وبعد كدا خد الاسم بتاع executable وباصيه لل output برضو في الخانة بتاعت " Name of executable " وهدوس Ok .

هروح بقا أدوس Start Debugging .. اللي بيكون على الشمال بيكون ال Registers بتاعت ال processor اللي هو بتاع ARM .. بيكون متاحلي إني أشوف ال modules أشوف ال memory أشوف أي حاجه أنا عايز أشوفها تقريباً .. أقدر أستخدم ال logic analyzer .. تقدر تشوف ال serial communication زي ال UART .

Volatile Type Qualifier

لو لاحظت في المثال اللي فات هتلاقي إن ال volatile هو اللي حل لينا المشكلة اللي كانت في المثال اللي فوق ده .. ال volatile ده بيخلي غصب عن ال optimization حتى لو هو مفتوح إنه ما يضيعليش piece of code أنا عايزه .

يبقا ال volatile ده keyword في ال C بنعمله apply مع ال variable .. عشان ده يقول لل compiler إن ال variable ده ممكن يتغير في أي وقت من غير ما يكون ليه حاجه تغيره في الكود .. يعني ايه الكلام ده ؟

بص هو ال optimization بتكون حساباته على مستوى الكود .. مش على مستوى ال microcontroller .. ف volatile بتقول لل compiler إن ال variable ده ممكن يتغير at any time بس مش من الكود اللي انت كاتبه .. يعني مثلاً ممكن يتغير من module موجود في ال Hardware تحت .

ال volatile ده بحطه قبل أي variable أنا خايف عليه .. ال variable ده أنا بستخدمه عشان أعمل access لحاجه من ال microcontroller أو أقرا حاجه منه .

Proper Use of C’s volatile Keyword

1- Memory-mapped peripheral registers

أول مثال على الحاجات اللي تتعمل volatile هي ال Memory-mapped peripheral registers :

يعني عندك ال CPU وانت روحت على ال Bus وفيهmodules على ال Bus وعنده registers R1, R2, ... etc .. وعندك كمان على ال bus فيه ال memory بتاعتك .. فانت عايز تستخدم ال volatile عشان من خلاله R1 ده معمول ان انت تقرا منه .

طب هقرا منه ازاي ؟ .. أنا هعمل pointer بيشاور على ال address بتاع ال Register ده اللي هو R1 فمثلاً ال pointer ممكن يكون كدا :

```
*13 == p ;
```

ال optimization كدا ممكن يشوف pointer بيشاور على address وهو كتب على ال address ده value .. يبقا انت منين بتقول ان انت ممكن وانت بتقرا منه تلاقي ان فيه قيمة اتغيرت ؟ .. ال optimization أوتوماتيكالي بيقولك عمر ما القيمة تتغير لأن انت already مكتبتش عليها حاجه فيقوم ماسحهولك .. فحاجه زي كدا بتخيلي أعمل volatile على ال pointer ده .. عشان أقوله خلي بالك المكان اللي بشاور عليه ده ممكن يتغير في أي وقت .. لأن ال Register ده موجود في Embedded Module .. وال Circuit بتاعت ال Peripheral ده ممكن تقعد تغيره في أي وقت .

2- Global variables modified by an interrupt service routine .

. Global variables accessed by multiple tasks within a multi-threaded application -3

ملحوظة : لو كتبنا ال volatile قبل ال int أو بعدها فعادي الاتنين شغالين مفيش مشاكل .

خلي بالك برضو :

* int const

ده معناه إن المكان اللي بيشاور عليه ال pointer ده هو اللي const .. لكن لو قولنا :

int * const

فده معناه إن ال pointer بيشاور على المكان ده بس وما ينفعش يشاور على مكان غيره .

نفس الكلام مع ال volatile .. لو ال volatile قبل ال * .. يبقا ال pointer ده بيشاور على volatile .

خلي بالك حتى من اسمه .. volatile دي يعني متطايره .. يعني ايه ؟ .. يعني متغيرة .. عشان متنساش .

فلما تكون قبل ال * معناها إن ال pointer بيشاور على داتا هي اللي متغيرة .. يعني ممكن تتغير at any time .

لكن لو أنا عملت الحركة دي بقا :

; int * volatile p

فدي معناها إن ال pointer هو اللي متغير .. بس بيشاور على حاجه مش متغيرة .

Peripheral Registers

دي حاجه كدا زي ال status register .. فدي مش انت اللي بتغيرها .. هي بتتغير automatic من ال embedded .

Incorrect implementation

```
uint8_t * pReg = ( uint8_t * ) 0x1234 ; // Wait for register to become non-zero
while ( *pReg == 0 ) { } // Do something else
```

احنا هنا أهو pReg ده بيشاور على مكان .. طبعاً احنا في الآخر المكان ده جوا module .. واللي بيشاور عليه ده ال status مثلاً .. واحنا دخلناه في while وبنقوله طول ما هو ب 0 ما تعملش حاجه .. لأن بيكون لسا ال module لنفترض إنه ال UART ده ما بعتش .. فييجي ال optimization يعمل ايه ؟ .. يقولك انت بتدخله في while وانت اصلا بتشاور على المكان ده 0x1234 والمكان ده لقيناه ان انت شايف ب 0 في الأول .. وفي الكود بتاعك عمرك ما روحت انك تكتب عليه .. يبقا عمره ما هيطلع برقم تاني .. يعني في الآخر هيفضل ب 0 .. يعني في الآخر هيفضل في while فراح يعملك ال assembly اللي طلع على أساس كدا .. فهو مش هيطلع assembly يروح يقرأ من المكان ده 0x1234 .. لأن هو ال optimization قاله ان انت مش هتغيره .. فيعملك assembly كدا :

```
mov ptr , #0x1234 mov a , @ptr loop : bz loop
```

هنا أهو هو عمل label اسمه loop .. ال label ده عامل زي الفانكشن في ال C .. بس في ال assembly بنسميه section .. وبيقولك فيه أهو branch لو هو ب 0 .. ولو لاحظت فهو في الأول خالص أهو راح في المكان ولما راح للمكان حط ال address اللي هو 0x1234 # في ال ptr .. وبعد كدا قاله روح لل address اللي بيشاور عليه ptr ده خزن القيمة في a .. القيمة دي هتطلع ب 0 .. وبعد كدا دخل loop وبيقوله طول ما هي 0 اعمل branch لل section اللي اسمه loop وكدا هيدخل في infinite loop .

فال optimization عمل معاك حركة وحشة في المثال اللي فات وقالك بدل ما كل شوية تروح تشوف المكان اللي في ال memory ده بقا فيه كام .. طالما انت مغيرتهاش في الكود يبقا مش لازم كل مرة تروح للمكان ده .. فال UART مثلاً ممكن يبعت وال Register يرفع ب 1 لكن انت لسا عمال تلف في ال infinite loop .. فدي طبعاً مشكلة .

طب حلها ايه ؟

حاجه زي دي بقا أول مانت تحط ال volatile للداتا للي انت بتشاور عليها .. بتقوله خلي بالك الداتا اللي أنا بشاور عليها هي اللي متغيرة .. فده هيبقى بال C كدا :

```
; uint8_t volatile * pReg = ( uint8_t volatile * ) 0x1234
```

وده هيكون ال assembly اللي هو هيطلعه :

```
mov ptr , #0x1234 loop : mov a, @ptr bz loop
```

فهنا في ال assembly لما دخل في ال loop .. قاله أول حاجه mov ال value اللي انت بتشاور عليها حطها في a .. ولو كانت ال a دي ب 0 اعمل branch تاني جوا ال loop .. كدا بقا لو فيه حاجه اتبعتت على ال Register ده فأنا هقدر أكتشفها .

يبقى انت كدا بقا تريح دماغك في عالم ال Embedded .. يعني ايه ؟ .. يعني أي Register هنروح نكتب عليه أو نقرأ منه .. نعمل pointer ل volatile data .. ودي قاعدة عامة إن أي حد بيحاول ي access ال peripheral registers بيعرّف pointer ل volatile data .

Interrupt Service Routines

انت عندك Global variable وعندك ال main وعندك function اسمها ال ISR .. وانت جوا ال while عمال بت loop على ال variable ده .. خلي بالك .. يعني ايه Interrupt service routine ؟ .. يعني انت عمرك ما روحت في الكود بتاعك ندهت على ال ISR ده .. امال ايه اللي حصل ؟ .. اللي بينده عليها هو ال processor .

ال processor لوحده Automatic لما يجيله interrupt بيسيب اللي في ايديه ويروح ي handle ال interrupt ده فيروح لل ISR .. فبالنسبة لل optimizer هو شايف إن ال variable ده عمره ما هيتغير .. ليه ؟ .. لأن انت عمرك ما دخلت جوا ال Function دي .. لأن انت عمرك ما ناديتها .. يبقا عمرك ما هتغيره .. فحاجه زي دي نحلها بإن احنا أصلاً نعرف ال variable ده من نوع volatile فحتى الجزء اللي احنا هنستخدمه جوا ال ISR ما يتمسحش هو كمان .

لأن ال optimizer كان ممكن يسمح ال ISR .. بس لو ال variable اللي بيستخدمه ال ISR ده كان متعرف إن هو volatile فأي حاجه بتستخدمه ال optimizer مش هيقدر ييجي جنبها .

Multi-threaded applications

انت عندك task1 و task2 وعندك Operating system .. ال OS ده شغال في 1) while) وبيعتمد على timer في ال Hardware .. ال timer اللي في ال hardware ده عمال يعد .. كل ما يوصل لعدات معينة .. بيبعت حاجه اسمها tick .. ودي يعني interrupt .. فال interrupt ده هيخلي ال OS يروح لل ISR بتاعه .. فال ISR بتاعه هيخليه يبدأ ي switch أو يرفع flag كدا من خلاله هيروح يندە ل task1 أو task2 .

نيجي بقا بالنسبة للكود .. الكود أو ال Optimizer ما يعرفش إن فيه timer هيعد وهيطلع ISR عشان يروح يندە على ال ISR ويروح يرفع flag تندە منه ال task .. فهو بالنسباله إن task1 ده مش هيتندە فممكن يمسحلك variables ما بينهم .

فالحل ايه ؟ .. الحل إن انت تخلي ال variable اللي جوا ال task من نوع volatile .

How lets see how to access the register absolute address

خلاص بقا احنا بتوع ايمبدد .. يبقا احنا عايزين نتعلم ازاي نكلم ال registers .. ازاي ن configure module عن طريق ان احنا ن configure ال registers بتاعته .

نفترض ان احنا عندنا register اسمه SIU واحنا عايزين نكتب عليه 0xFFFFFFFF وال absolute address بتاعه هو ده 0x30610000 .. وال Register ده بيتكون من 32 bit .. يعني 4 byte .. وكل بايت بيعبر عن Port معين .

يعني لو قولنا ان ده module اسمه GPIO فهو عنده : PORTA, PORTB, PORTC, PORTD .. وكل Port فيهم فيه 8 pins من 0 ل 7 .. فاحنا عايزين نكتب عليه نعمل ايه ؟

Solution 1

Declare a pointer -1

; volatile int *p

أنا هعمل volatile int لأن أنا عايز أشاور على حاجه في ال memory .. هنا بقا ما تقوليش ال memory يعني RAM أو ROM .. خلي بالك إن ال processor ده بي access bus وال bus ده بي access modules كتير .. فاحنا كل module ليه base address وال size بتاعه .. فبالنسبة لينا ال Bus دا كله بنسميه Memory bus range أو Memory address range .. فده معنى ال Memory اللي نقصده في الايمبدد .

Assign the address of the I/O memory location to the pointer -2

; p = (volatile int *) 0x30610000

هنا خلينا ال p ده يشاور على ال address اللي احنا عايزين ن access عليه .. وطبعاً عملنا casting ليه عشان نقول لل compiler ان ده مش Numerical value إنما ده Address .. احنا خدنا الكلام ده في ال Lecture بتاعت ال pointers .

Output a 32-bit value -3

; p = 0xFFFFFFFF*

ده كدا طريقة الاستكنياص .. يعني أسهل طريقة وتخلصني .. تعالى نشوف طريقة professional شوية .

Solution 2

بيقولك ان الطريقة اللي فاتت دي مش حلوة .. ليه ؟ .. انت عرفت p وروحت قولت p بيشاور على ال address ده وبعد كدا روح اكتب على ال p .. فانت كدا عملت 3 خطوات .. فانت متخيل بقا لو انت بتتكلم على 20 أو 30 module وفيه modules بال 40 register .. فمش هينفع تقعد تعملي كدا مش هتبقا professional .. فتعالى نشوف الحل ده بقا .

; volatile unsigned long *) (0x30610000)) = 0xFFFFFFFF))*

السطر ده معناه ايه بقا ؟ .. انت بتقوله ال pointer ده اللي هو volatile unsigned long اللي بيشاور على ال address ده اللي هو 0x30610000 اعمل حواليهم قوسين وحط جواه بال * اللي برا خالص دي ال value اللي عندك دي اللي هي 0xFFFFFFFF .

أو ممكن نكتبها بالطريقة دي :

#define MYREGISTER *((volatile unsigned long*)(0x30610000

فلما ييجي في الكود بقا يقول MYREGISTER = 12 .. يحصل إن ال Pre-compile يشيل ال MYREGISTER ويعمل text replacement يحط مكانها المنظر ده ويساويه ب 12 .

طب ده ميزته ايه ؟

إن انت مش معرف variable فمش واخد من ال memory مكان عشان تسيفه .. ده انت كل اللي بتعمله فيه .. إنك بتقوله شايف ال address ده ؟ .. اه .. حط جواه ال value دي .. بس كدا .

فدي بتترجم علطول ل assembly .. فمفيش بقا memory هيروح ليها يشوف القيمة اللي شايلها ويروحلي جواها وهكذا .. فهي دي الطريقة ال professional اللي 80% من الناس بتوع ال Embedded بيشتغلوها .. إنهم يعملوا Macros .

أنا عامة بحب المنظر ده بس فيه أوقات بنحتاج نعمل منظر تاني واللي هو المنظر الجاي ده .

Solution 3

```
typedef union { vuint32_t ALL_ports ; struct { vuint32_t PORTA : 8 ; vuint32_t PORTB : 8 ; vuint32_t PORTC : 8 ; vuint32_t PORTD : 8 ; } SIU_fields ; } SIU_R ;

; volatile SIU_R* PORTS = ( volatile SIU_R* ) 0x30610000
```

احنا قولنا قبل كدا ان فيه registers بيكون فيها كل bit بتعبر عن حاجه .. وعشان كدا احنا خدنا ال bit field جوا ال lecture بتاعت ال structure .. احنا خدنا ال bit field options وقولنا ان ده وظيفته ان احنا نقدر ن handle كل مجموعة bits على حدة .. فانت دلوقتي عندك register بيتكون من 4 byte وكل byte بيعبر عن Port .. طب ما تخليك professional بقا .. اعمل typedef ل union .. ليه union ؟ عشان يبقا shared memory .. ال union اللي انت معرفه ده هيبقا 4 byte بس .. عشان هو shared memory .. يعني ال 4 bytes دول هما shared ما بين حاجتين .. ALL_ports ده member من نوع integer يعني بيشيل 4 byte فال 4 byte بتوع ال union هما shared لل ALL_ports دي .. في نفس الوقت أنا عملت struct لل member التاني .. هو اسمه SIU_fields .. ال member ده بيتكون هو كمان من unsigned integer .. ولو انت فاكر في ال bit field فاللي احنا عاملينه فوق ده بنقوله أول حاجه احجزلي ال 8 bit الأولانيين .. ثم التانيين .. ثم التالتين .. ثم الرابعين .. فبقا ال member الأول في ال struct بيشاور على ال أول 8 bit .. وتاني member في ال struct بيشاور على تاني 8 bit (في ال union) .. وتالت member في ال struct بيشاور على تالت 8 bit ورابع member في ال struct بيشاور على رابع 8 bit .. وال members دول اسمهم بالترتيب هنا PORTA, PORTB, PORTC, PORTD .. وفي الآخر بقا كل دا على بعضه ال struct بال union بقا اسمهم SIU_R .

الموضوع ده بقا اداني ميزة حلوة قوي .. لو أنا عايز أكتب على ال Register اللي بيتكون من 4 byte مرة واحدة .. يعني كإني بكتب على ال 4 ports مرة واحدة .. كإني بطلع High 32 على Pin 32 في مرة واحدة فهقوله كدا :

```
; PORTS -> ALL_ports = 0xFFFFFFFF ;
```

أو لو أنا مثلاً عايز أكتب على Port واحد بس ولنقل إنه PORTA فهنعمل كدا :

```
; PORTS -> SIU_fields . PORTA = 0xFF ;
```

فدي طريقة جميلة بتريح الدماغ وتخليك لو انت عندك Register بقا بيتكون من كزا حاجه .. ما تقعدش بقا تعمل لكل واحد shift .. لا انت قول اسم ال register و اكسيس بال dot وريح دماغك .

يبقى الكود اللي احنا عملناه فوق ده انت تعرفه مرة واحدة بس في ال header وخلاص .. وتستخدمه بعد كدا في الكود بتاعك والكود هيبقا more readable والدنيا peace .

Toggle LED on STM32F103CX

عشان تتعامل مع حاجه في GPIO معين .. لازم ان انت تفتح ال gpio ده .. طب تفتحه ازاي ؟ .. لازم تخلي ال RCC تطلع أو ت Enable ال clock بتاعتها اللي هي داخلة لل gpio لأن هي by default هي مقفولة .. تاني حاجه محتاج أبقا عارف ال Base address .

فيه في proteus فوق في ال bar موجود tab اسمها Debug ودي بتساعدك تشوف أي module انت عايزه .. السهم اللي موجود تحت جنب سهم التشغيل ده بيستخدم في ال Debugging في proteus .

Very Important Question

الأسئلة دي بتكون مهمة قوي وبتقيس بيها بقا حتة انت فاهم ال pointers مع ال volatile مع ال const ولا لأ .

Usage of const and volatile together

عندنا مثال بيقولك انت عندك processor بيشاور على register .. ال register ده RW وال offset بيكون 0 وال access بتاعه Read/Write .. وعندك reg2 هو Read-only access يعني تقدر تقرأ منه بس لكن ما تقدرش تكتب عليه .

لو أنا عملت كدا :

```
; uint32_t volatile * const Preg1 = ( uint32_t ) 0xFFFF0000
```

ده معناه إن ال pointer هو اللي constant يعني هو هيفضل يشاور على ال Address ده علطول .. فمينفعش أقوله Preg++ .

ال volatile بقا مستخدمينه هنا عشان ده بقول فيها ان الداتا للي أنا ب point عليها ممكن تتغير قدام بس مش من الكود .. فبقول لل optimization ما تعملش فيها حاجه .. خلاص الحمد لله احنا فهمنا الحتة دي وهنسد معاها في الانترفيو كويس قوي .

طب ايه الجديد بقا ؟

; unsigned const volatile * const Preg2 = (uint32_t) 0xFFFF0004

ده معناه إن أنا عندي constant pointer بيشاور على constant و volatile داتا .. وده بقا اللي بيلعب بيه معاك في الانترفيو .. const مع volatile ازاي ؟

قوله خلي بالك هي const بالنسبة لل compiler .. يعني أنا ك programmer لو جيت أغيره فال compiler هيمنعني .. volatile معناها إن هو ممكن يتغير من برا .

وهنا في المثال بتاعنا بقا (وعموماً أي register بيكون read-only هو اللي هنعمل معاه كدا) .. ال reg2 ده واخد read-only access من ال Hardware يعني انت تقدر تقراه بس .. يعني لما تيجي تكتب كود C فلو سمحت اقراه ما تكتبش عليه .. فأنا وأنا بكتب ال header بتاعي .. أول ما ألاقيه read-only هخاف إن أنا أنسى وأكتب عليه .. فهخليه const .. فكدا لو نسيت وجيت كتبت عليه هييجي ال compiler يمنعني فأعرف ساعتها .

Bare metal Embedded SW

[Most of the interview questions are concentrated here]

بص انت لما تيجي تتعين ان شاء الله .. انت احتمال كبير مش هتشتغل على بوردة معروفة .. فيه بعض البوردات بتبقا انت بتنزلها ال IDE .. فممكن يكون فيه IDE حلو قوي بس بفلوس كتير قوي .. فانت ممكن تشتغل من غيره .. فيقولك ايه .. انت عايز Editor ؟ .. اه .. نزل Notepad أو Sublime .. أو أي Editor على النت For free .

تاني حاجه انت عشان ت compile محتاج ايه ؟

هتقولي والله محتاج toolchain .. طب ايه هي ال toolchain ؟ .. دي اللي بيكون فيها ال GCC وفيها ال libraries وفيها ال Debugger وفيها ال binary utilities .

ايه تاني ؟ .. تقوله أنا كدا خلاص هبدأ أحط ال C code بتاعي في ال Application .. بس ال application لوحده مش هيشتغل .. طب محتاج ايه ؟ .. محتاج startup ومحتاج linker .. ومحتاج إن أنا أعمل Automation لل build دا كله ب Make file .

يقولك طب لو انت خلاص عملت كل الكلام ده .. تقدر ت compile وت debug من غير ال IDE اللي بفلوس ده ؟ .. اه .

طب ال IDE بيعمل ايه ؟

بيقولك ال IDE لما بييجي ي build هو بيروح يندە في الآخر على Make file هو كاتبه .. ولما ييجي ي compile بيستخدم ال gcc بتاع ال toolchain واللي انت ممكن تجيبه لوحدك .. ولما ييجي ي debug بيستخدم ال gdb برضو بتاع ال debug اللي انت كنت ممكن تنزله لوحدك .. طب ما الدنيا بيس أهي .. طب ازاي بقا نعمل الكلام ده ؟

How to write Embedded C Code From Scratch without IDE ?

عشان نعمل حاجه زي كدا يبقا احنا محتاجين نتعلم :

1- Cross ToolChain

2- Make File

3- C Code files

4- Linker Script

5- Startup.s

كل واحده بتعمل ايه وازاي نكتبها وازاي نقراها وايه الميزات بتاعتها والكلام ده كله .

احنا النهاردة هناخد بس ال Cross Toolchain .. المرة الجاية هناخد ال Make File وال linker وال startup .. والمرة اللي بعديها هناخد ال Debugger وال binary utilities .

OS App Vs Bare metal SW

ال Bare-metal application هو في الآخر ال C Code بتاعك اللي بيكلم ال Hardware أو ال HWLibs ودي يعني ال Drivers اللي انت كاتبها .. يعني من الآخر انت بتكلم ال Hardware دايركت .. يعني اللي بيكتب bare-metal application لازم يكون Embedded software engineer .

ال OS بقا لو قولنا مثلاً إننا شغالين Linux .. فال Linux ليه system call .. أنا ك application بكلم ال system call .. وال OS بقا هو اللي بيتعامل مع ال Hardware .. فهو كدا بي abstract ال application عن ال Driver وال Driver هو اللي يكلم ال Hardware بقا .. يبقا انت في الحالة دي مش لازم تكون مهندس ايمبدد .. ليه ؟ .. لأن أنا مش محتاج هنا إن أنا أفتح ال SPECs بتاعت ال Hardware وأتعامل مع ال Registers .

Cross-compiling toolchains

أول حاجه يعني ايه toolchain ؟ .. يقولك toolchain دي يعني compiler + assembler + linker + debugger + libraries .. ما هي ال libraries ؟ .. هي C Code معمول ليه compile بال architecture اللي هي run عليه .. وكمان ال toolchain بيكون فيها number of helper programs ودول اللي هما ال binary utilities .. يعني الحاجات اللي بتستخدمها عشان تعمل analysis لل binary بتاعك .

ال Debugger بنستخدمه عشان يتوصل بال Debugger Circuit اللي بتخلينا ن debug .. وال Compiler مع ال linker بيطلعلي ال program file اللي بحرقه على البوردة عشان أشغلها .. وال libraries اللي بعمل link معاها .

بيقولك لو انت شغال على PC زي redhat أو Ubuntu أو Linux .. بتلاقي ال toolchain native .. ايه هي ال native toolchain دي ؟ .. هنعرفها بعد شوية .

يبقا دلوقتي toolchain يعني :

compiler - Binary Utilities - Debugger - Kernel Header (related to Linux) - C/C++ Libraries - (Trace and profile (related to embedded linux .

ال native toolchain بتكون هي ال toolchain اللي انت مسطبها على الكمبيوتر بتاعك .. فيه فرق ما بين حاجتين ال host وال target .. ال host يعني الجهاز اللي انت شغال عليه دلوقتي وبت develop عليه .. ال target هو الجهاز اللي انت مطلع binary يشتغل عليه .

خلي بالك ال toolchain عشان تشتغل لازم تتسطب عندك على الكمبيوتر .. يعني ت run عندك على الكمبيوتر يعني ت run على ال intel processor بتاعك .. فانت في الآخر لما تقول ان ال toolchain دي انت نزلتها وشغلتها عندك على الكمبيوتر فده يعني ايه ؟ .. يعني بت run على ال intel أو ال AMD Processor بتاع ال PC بتاعك .. حلو قوي .. لما انت عملت Compile لل C project بتاعك وطلعت binary وال binary ده اشتغل على نفس الكمبيوتر اللي انت عملتله build .. يعني ال host وال target كانوا واحد اللي هو Intel في الآخر .. يبقا ده بنسميه native toolchain ودي ال toolchain اللي بتتسطب على architecture وتطلع binary ي run on the same architecture .

فاكر ال MINGW اللي انت نزلته في أول الكورس بتاع ال C ؟ .. انت نزلته على الكمبيوتر بتاعك ولما طلعت binary وشغلته اشتغل على الكمبيوتر بتاعك .. يبقا ده بنسميه native toolchain .

طب ال cross toolchain بقا ؟ .. هو ده اللي بنعمله دلوقتي زي ال STM32CubeIDE و Keil Uvision

---

فبييجي في بعض ال MCQ في شركات في الامتحان بتاعها يقولك هو ال executable اللي طالع من ال cross toolchain ده ينفع نعمله run على نفس ال host machine ؟ .. هتقوله لأ طبعاً لأنه بيطلع executable اه بس for target machine .. ال architecture اللي أنا ه run عليه مش اللي أنا already مسطب عليه ال toolchain .

أنا عارف ان ال ARM ده اللي هينزل هو ال toolchain بتاعته هتتسطب عندك على Intel بس هتطلعلي binary ي run على ARM فدي تبقا ARM Cross toolchain .

لو بصينا بقا على ال Build tools هتلاقي :

1- gcc

وده كدا ال compiler بتاعك

2- ld

وده كدا ال linker بتاعك

3- make

ده ال make command بيستخدم عشان تنده على ال make file اللي كتب فيه ال process بتاعت ال compilation .

ar -4

ده ال archive وده اللي بيطلعلك حاجه اسمها static library هنشوفها دلوقتي .

لو بصينا بقا على ال Binary Utilities هتلاقي :

readelf -1

دي بتخليني أقدر أطلع information من ال executable file بتاعي

objdump -2

دي بتخليني أحول ال executable file بتاعي اللي طالع ده ل assembly عشان أتفرج عليه

nm -3

ده binary utility موجود كدا في ال toolchain عشان تخليني أقرا ال symbols من ال symbols table اللي موجودة في ال binary file . (لو الكلام ده انت مستغرب منه ما تقلقش .. هتفهمه المحاضرة الجاية ان شاء الله) .

strings -4

دي عشان تطلع أي string مكتوب في ال code .

strip -5

ده بيستخدم عشان يطلع جزء من ال section .. يعني انت عايز تطلع ال binary بس من ال executable .. يعني انت مش عايز أي debug .. انت عايز تطلع ال binary بس فساعتها بنستخدم ال strip ده .

addr2line -6

دي بقا جميلة قوي .. دي انت مثلاً تقوله ايه .. بتديله ال executable file اللي طلع منك وتديله address .. يعني مثلاً تقوله عند address مثلاً 0x10000 .. هو هيشوف ال address ده في الكود عنده .. ايه ال instruction اللي موجود وال instruction ده موجود في أنهي file في أنهي سطر c.

يعني مثلاً وانت بت debug على البوردة وال debugger قالك ان انت حصلك crash في ال code flash عند address 10000 .. فانت تاخد ال address اللي هو 0x10000 وباستخدام ال binary utility دي تطلع ال address ده في سطر assembly ايه والسطر ال assembly ايه ده بيعبر عن file.c و line كام في الفايل ده .. فتروح تشوف المشكلة في السطر ده ايه بالظبط .

size -7

دي بتجيبلك كل section موجود في ال executable بتاعك وتقولك هو واخد size قد ايه .. يعني مثلاً ال data section ال txt. واخده قد ايه .. ال debug section واخد قد ايه .. وهكذا

ال gdb ده عشان ت debug ال software اللي بي run على البوردة .

Definition

الشركات اللي أكل عيشها إنها تعمل toolchain هي بتكتب ال toolchain ك source code وفيه بعض الشركات ال source code بتاعها موجود على النت .

ال Build system ده بقا هو اللي بيكون فيه ال source code بتاع ال gcc نفسه بتاع ال toolchain وبنعمله compile عن طريق ال GCC native بتاع ال host machine .. عشان أحطه على ال host machine بتاعتي وأعملهم الاتنين cross compile عشان يطلعلي ال executable file بتاعي يقدر ي run على التارجت اللي هو البوردة أو ال microcontroller بتاعي .

احنا ما يمهناش ان احنا ن build ال toolchain أصلاً .. الخطوة دي أصلاً هما الشركات منزلين ال source code بتاع مثلاً ال ARM GCC .. احنا بنشتغل بال GCC-Cross compile علطول جاهزة .. طب نجيبه منين ARM ده ؟

أنا هحطلكم لينك ازاي تنزلوه .

Components - GCC

بص احنا من أول هنا بقا هنبدأ نشتغل بال commands .. تعالى بقا معايا

أول حاجه .. عشان نفتح اي فايل موجود في ال directory اللي احنا واقفين فيه ونعرضه قدامنا بداخل ال Mingw فهنكتب كدا :

/.

وتكتب بعدها اسم الفايل اللي انت عايز تفتحه .

مثلاً انت عايز تشغل ال compiler اللي هو gcc إنه يعمل compile ل main.c ويطلع منها main.exe فهتكتب كدا :

gcc.exe main.c -o main.exe/.

خلي بالك ال o- دي معناها طلعلي output كزا يعني .

ولو انت عملت الكلام اللي فوق ده من غير ما تقوله o-

gcc.exe main.c/.

ففي الحالة دي هيطلعلك file اسمه a.exe

فال o- دي تقريباً كدا بتخليك تحدد اسم الفايل اللي انت عايز تطلعه في ال output .

طب لو أنا عايز أقوله اعمل Compile لل main.c و include معايا header أنا عايزه .. فساعتها بتقوله I- وتديله اسم ال directory اللي فيها ال include بتاعتك .

لو أنا عايز أ Enable ال warning أثناء ال compilation process فساعتها هقوله Wall-

---

لو أنا عايز أقوله إن لو فيه warning يعملي error .. يعني ما يعديش الموضوع ويعمل compilation error فساعتها هنعمل Wall- وهنزود عليها Werror-

---

لو أنا عايز أديله options كتير بدل ما كل مرة أقعد أكتب السطور دي كلها .. ساعتها هقوله :

gcc main.c @option-file

بحيث ان option-file ده هو اسم الفايل اللي انت حاطط فيه ال options بتاعتك .

فيه برضو E- ودي معناها اعمل compile لحد بس ال pre-compile .. يعني اعمل pre-compile واقف على كدا .. وطلعلي ال main.i file .. وال main.i فايل ده بيشيل كل ال hash ويحط ال text replacement بتاعتها .

وقولنا S- بتقوله اعمل compile بعد ال pre-compile .. يعني طلعلي ال assembly واقف .

وقولنا ال C- بتقوله طلعلي ال object file واقف .

طب لو احنا عايزين نعرف كل ال commands دي نعرفها منين ؟

أبسط طريقة وأحسن طريقة إن انت تعمل كدا :

gcc —help

ده هتلاقيه كاتبلك فيه كل الحاجات اللي قولنا عليها فوق دي .

(Creating a Static Library (ar command

ايه بقا ال ar ده ؟ .. فيه حاجه اسمها static library وفيه حاجه اسمها dynamic library .. نفترض ان انت شركة كنت بتعمل driver بتاع ال CAN controller .. فانت عملت CAN.c وعملت CAN.h وكان ال driver ده 800 ورقة .. وقعدت تكتب فيه لمدة 3 .. 4 شهور .. فالشركة بتاعتك دي هتبيع ال driver ده لشركة تانية كبيرة قوي .. الشركة التانية دي عملت application محترم هي كمان .. وبعد ما عملت ال application ده اللي فيه فايلات كتير قوي .. محتاجه ان هي تكلم الدرايفر بتاعك عشان من خلاله تكلم ال CAN controller اللي هو بيكلم ال hardware عشان يبعت ويستقبل .. فلو انت اديت ليهم الفايلات c. وال h. فهو كدا خلاص خد ال source code بتاعك والكلام ده ما ينفعش .

فيقولك خلاص .. انت مثلاً عامل في ال c. ال Implementation بتاع function اسمها CAN_Init .. يعني دي اللي بت initialize ال CAN controller مثلاً .. تعالى في ال h. واعمل extern CAN_Init .. فانت كدا عملت ال prototype بتاعك .. انت هتدي للشركة التانية ايه بقا ؟ .. هتديها ال CAN.h فايل .. طب ليه ؟ .. عشان يعمله include معاه في ال application فيقدر يشوف ان فيه function اسمها CAN_Init وال application بتاعه لما ييجي يخش على ال compilation .. هييجي عند حتة ال object file .. فالمفروض ال binary code بتاع ال application بتاعه مشي كويس لحد النقطة دي لأنه متعرف فيه إن ال function دي نوعها extern .. فهي دي معناها ايه ؟ .. دي معناها يا compiler عدي الدنيا لأن ال linker هو اللي هيشوفها .. يعني انت شغلتك تطلعلي ال object وبعد كدا ال linker هيشوف الكلام ده كله ويربطه مع بعض .

فالمهم ال object اللي طلع من ال application ده هيخش على ال linker وهو محتاج مين بقا ؟ .. محتاج ال definition بتاع ال function اللي اسمها CAN_Init .. يقولك انت كشركة طلعلي static library واللي هي Lib.a ودي كإن فيها ال object type بتاع ال CAN.c .. فهيدخل ال Lib.a ده مع ال object file بتاع ال customer على ال linker يعملهم link مع بعض يطلع ال executable النهائي اللي بيشتغل .

سؤال انترفيو

هل ال Linker بياخد ال Library كلها ولا بياخد بس ال object بتاع الجزء المستخدم ؟ .. فهتقوله هو بياخد بس ال object بتاع الجزء المستخدم .

حاجه زي كدا ازاي نعملها بقا ؟

عشان تعمل library .. أول حاجه لازم تطلع ال object file بتاع ال library .. فهتكتب ده :

```
gcc.exe -c can.c -o can.o/.
```

كتبنا c- عشان ما يروحش يطلعلي ال executable والدنيا تضرب مني .

خلي بالك : الكلام اللي بنقوله ده مش مع ال Mingw بس .. لا دا تقدر تستخدمه مع أي toolchain في الدنيا .. ممكن نعمل static library بسهولة بس احنا بنشتغل بإيدينا عشان نفهم كل حاجه .

عشان تعمل static library هتكتب الكلام ده :

ar.exe rcs/.

وتديله ال object files وهو بياخدهم مع بعض ويعملهم library

ar.exe rcs lib_can.a can.o/.

بحيث ان lib_can.a ده اسم ال library أنا اللي محدده عادي .. وباصيت اهو ال object file الوحيد اللي عندي اللي هو can.o .. لو فيه غيره ممكن نضيفهم عادي بس حالياً ده اللي موجود .

وال library دي بقا هي دي اللي أنا هديها لل customer .. فانت مثلاً ممكن تعمل درايفر لل ATmega32 وتخليها library وتخليها عندك .. تقول دي ال library بتاعتي أنا .. يوم ما أنا استخدم ال ATmega32 .. ه link مع ال library دي .

عشان خاطر ت add another object هتعمل كدا :

ar r file3.o

يعني لو انت جالك فايل جديد مثلاً file3.o وعايز تزوده في ال library هتكتب ال command اللي فوق ده .

لو عايز تمسح object كان موجود في ال library هتباصيله دي :

ar d file3.o

بتحط اسم الفايل عادي .. أنا هنا عامل مثال بس .

لو عايز تشوف ال object files الموجودة جوا ال library هتعمل كدا :

ar t libmylib.a

ركز بقا في الحتة دي مهمة جداً .. لو عندك library وعايز ت extract منها ال object تديله x .. يعني لو انت قولتله

ar.exe x lib_can.a/.

فانت هتقولي ده كدا ممكن ال customer ي hack ال object files ويطلعها .. يعم هو طلعها .. هيعمل بيها ايه ؟ .. مش هيعمل بيها حاجه .. ما ال object ده عبارة عن ايه ؟ .. binary على بعضه .

خلي بالك بقا لو انت جيت تعمل كدا :

gcc.exe main.c -o main.exe/.

فده هيطلعلك error يا صاحبي .. ال compiler طلع main.o بس لما جه يعمل ليه link ملقاش ان فيه حاجه شايلة جواها ال definition بتاع ال CAN_Init وده اللي احنا متوقعينه لإن احنا لسا مباصاناهوش لل library .. طب دي نحلها ازاي ؟

وانت بتباصيله ال main.c باصيله معاها ال lib_can.a بتاعتك

gcc.exe main.c lib_can.a -o main.exe/.

فدي هتشتغل معاك ومش هتدي error .

خلي بالك : كل اللي فات ده وال terminal كان مفتوح في ال directory ده :

C/MinGw/bin/

بيقا كدا فهمنا ازاي جزء من الكود نعمله static library ونديه لأي حد زميلنا أو حد في المشروع ياخده ي link ه معاه يطلعله في الآخر ال executable النهائي .. وده اللي شغالين بيه في ال AUTOSAR .. ال AUTOSAR هو Layers كتير .. هو Modules .. فيه Application و RTE و BSW و OS و Diagnostic و COM Stack وليلة كبيرة قوي سعادتك .. وعمر ما فيه حد هيديله ال source code بتاع كل الكلام ده على الجهاز .. امال ايه اللي يحصل ؟ .. تيجي Valeo تكتب ال application .. تيجي الشركة اللي أصلاً شغلت Valeo مثلاً BMW تقولها هتكتبي انتي Application1 وأنا هكتب Application2 .. وال Interface ما بيننا كزا كزا .. وتيجي Valeo تشتري ال AUTOSAR Stack مثلاً من شركة Vector ومثلاً تشتري من Mentor ال OS بتاع ال AUTOSAR وتحطهم كلهم عندهم على ال Machine وت compile .. فهي مش بتاخد ال source code بتاع كل حاجه .. هي بتاخد ال library بتاعت كل module .

فبقت كل حاجه هي جزء header مع configuration مع library نجمعهم مع بعض .. ييجي ال linker يعمل link ليهم مع بعض يطلعلك في الآخر ال executable النهائي اللي بيتحط على البوردة .

Most used ARM Cross-toolchain

ايه اكتر الحاجات المستخدمة لل ARM ك Cross toolchain ؟

أكتر حاجه مستخدمة هو

(GNU GCC for ARM embedded Processors (free and open source

وده اللي هنشتغل بيه .

برضو فيه

(armcc from ATM ltd. (ships with KEIL, requires licensing

ده اللي موجود في ال Keil Uvision وده محتاج license .

طريقة تنزيل ال ARM toolchain دي موجودة في المحاضرة .. واحنا ممكن نشتغل الدبلومة كلها بال toolchain دي من غير IDE اصلاً .. ولو انت شايف انك كدا بترخم على نفسك .. هقولك خلاص استخدم ال toolchain دي مع ال Eclipse العادي خالص واشتغل أي كود انت عايزه سواء Tiva C أو STM32 أو أي حاجه .

أنا عايز مخك يبقا كبير .. ما تخليش مخك ي focus على بوردة واحدة أو على toolchain واحدة .. ال embedded سهل وبسيط والدنيا فيه لو انت فهمتها من تحت كويس هتلاقي الدنيا بسيطة خالص .

ولو انت عايز تشتغل AVR هتنزل ال AVR toolchain وتحطها على الكمبيوتر وتفتحها من Eclipse وتشتغل AVR على Eclipse .. ايه ده يعني Eclipse ممكن يشتغل لل host ولل ARM ولل Atmel ؟

اه ولو فيه Power PC ممكن يشتغل كدا كمان .. أصل الموضوع بسيط .. وهو مش عشان Eclipse حلو .. هو Eclipse في الآخر IDE .. هو الفكرة كلها انت نزلت ال toolchain ك cross وسطبتها عندك على الجهاز ؟ .. ممكن تستخدمها direct من غير أي IDE أو ممكن تستخدمها ب IDE .

ممكن في مرة أستخدم Eclipse مع ARM .. ممكن في مرة أكتب علطول مع toolchain من غير أي IDE .. ممكن مرة تانية أستخدم ال CubeIDE .. ممكن مرة رابعة أستخدم ال Keil Uvision .. انت بقا خلاص خلي مخك أكبر من الكلام ده كله .

طب خلاص أنا نزلتها ال toolchain وسطبتها وبقت عندي .. لما أجي أفتحها بقا هلاقي ايه ؟ .. هتخش على ال bin واللي هي اختصار ل binary .. بص بقا هتلاقي ايه

-arm-none-eabi/.

وبتكتب بقا بعدها اللي انت عايزه تستخدمه .. يعني مثلاً لو هتستخدم ال compiler هتكتب كدا :

arm-none-eabi-gcc/.

ولو هتستخدم ال linker هتكتب كدا :

arm-none-eabi-ld/.

ولو هتستخدم ال archive اللي هو بتاع ال library هتكتب كدا :

arm-none-eabi-ar.exe/.

خلي بالك بقا دا كله ل ARM لإن أنا منزل ARM toolchain .

سؤال في فاليو

فيه بقا toolchain تانية ل ARM بس بدل ما اسمها arm-none-eabi .. لأ هي اسمها arm-linux-gnuapi- .. طب ايه الفرق ما بين ده وده ؟

هو arm-none-eabi ده بيستخدم مع ال bare-metal applications .. يعني انت بتكتب كود بي run على ال ARM دايركت .. لكن arm-linux-gnuapi بيكون فيه حاجات أزيد لأن هو ال application مش هيكلم ال microcontroller دايركت .. دا هيكلم ال linux اللي هي على ال ARM Microcontroller .

يعني لو انت جبت ARM Board وحطيت عليها Embedded Linux وعايز تبني application فوق ال linux ف arm-linux-gnuapi هيسد معاك في الموضوع ده .. لكن arm-none-eabil دي لو انت هتكتب bare-metal دايركت على ال Microcontroller بتاعك .

فالسؤال كان ده كدا .. قاله مين اللي بيستخدم عشان تكتب bare-metal application على ARM Board .. هل هو arm-none-eabi ولا arm-linux-gnuapi .. فساعتها هتختار arm-none-eabi .

كل ال commands اللي احنا خدناها من شوية هي applicable هنا بقا .. الدنيا بسيطة ما تقلقش .

Compilation Process

[Most of the interview questions are concentrated here]

ايه هو بقا ال compilation process ؟

أول حاجه عندي ال app.c ده ال application بتاعي .. هتكتب ال bare-metal application يعني ايه ؟ يعني ال executable file النهائي ده هيتحط في ال Flash memory بيكلم ال hardware Direct .. ال application هيخش على ال compiler مع ال startup وهيطلع من ال Compiler ال o. فايل والاتنين دول هيتعملهم link مع بعض مع ال libraries .. وال linker script ده هيديهم تعليمات وهيطلع لينا في الآخر ال executable file .. وال executable ده بنجيب programmer و debugger سيركت عشان نحرقه .. ال binary ده نحطه على البوردة .. وهناخد برضو session نتعرف فيها على كل أنواع ال Debugger وال Programmer Circuit اللي موجودة في ال Embedded .

أول حاجه عندي هي ال Pre-processing stage .. ايه اللي بيحصل فيها ؟ بيدخلها app.c يطلع app.i فايل .. طب ده بيعمل ايه ؟ .. ده بيروح لكل ال Hash اللي هما ال Directives ويعملها resolved .. يعني يشيلها ويحط المقابل ليها .

بعد كدا بقا ال app.i ده بيخش على ال code generation stage وده اللي هو ال compiler نفسه وفي المرحلة دي بقا بيحصل ده :

Higher level language c code will be converted into processor architecture level assembly

---

خلي بالك بقا ان المرحلة دي هي based على ال architecture .. يعني كل architecture ليه his own assembly .. ال ARM ليه assembly .. ال powerPC ليه assembly .. كل واحد ليه ال assembly بتاعه .. فال code generation stage دي فيه بعض المراجع تقولك ان ده ال compiler .. هو ده اللي بياخد ال i. فايل ويطلعلي ال assembly code .

بعد كدا بقا ال assembly code ده يروح لل assembler stage وده بقا اللي بي convert كل سطر assembly لل opcodes وال operands بتاعته .. فهو بيطلع من ال assembly ده ال binary اللي مكافيء لل architecture ده .. وده اللي بنسميه object file .. ومن هنا بقا ركز على الاسم اللي جاي ده اللي هو Relocatable object file .

سؤال انترفيو

ايه هو ال Relocatable file ؟

نفترض مثلاً ان احنا عندنا ده

```
add R1, R2, R3
```

الكلام ده عند Address كام ؟ يقولك ال Address مثلاً هنا 0x12 .. طب هو فيه Address كدا في الميكروكنترولر ؟ تيجي تبص في الداتا شيت متلاقيش الكلام ده .. فال Address ده جه منين ؟ .. فيقولك ان ال Addresses دي هي Virtual .. يعني هي وهمية طلعت وخلاص ومستني ال linker هو اللي بعد كدا ياخد ال assembly ده ويطلع فايل فيه ال Addresses بتاعت ال physical .. يعني بي map ال physical address بال virtual address المقابل ليه على الميكروكنترولر .

يبقى ال Relocatable object file هو عبارة عن ايه ؟ .. دي object أو binary كود لسا مش بي map ل physical address في ال memory .. هو virtual address .. عشان كدا بنقول relocatable .. هو لسا مش located correctly according to the physical memory .

مين اللي مسؤول ان بعد كدا ياخد السكاشن أو ال objects ويحطهم في فايل واحد ويظبط ال addresses بتاعت الفايل ده ان هي تنادي ال RAM بتاعت ال Microcontroller هو ال linker script .. وده سؤال جه في Valeo قبل كدا .

يبقى ال assembler stage ده طلعلي ال app.o وده processor architecture specific machine codes with (no absolute address (relocatable .

بعد كدا بقا ال app.o يخش على ال linker مع ال libraries ي link مع بعض يطلعوا executable file مع debug information .. هنا بقا مفيش حتة virtual addresses .. لا خلاص طلع فايل فيه ال Addresses بتاعت ال memory الحقيقية .. ال linker script ظبط الحتة دي خلاص .

أنا مش محتاج ال debug sections أنا محتاج سكاشن معينة .. فبدخله على strip أو objcopy في ال binary utilities بقدر أطلع منه ال bin. وال hex. اللي شايلين بس فيهم السكاشن المهمة وشايل السكاشن اللي ليها علاقة بال debug .. خدت بعد كدا ال binary أو ال hex ده دخلته على ال Programmer circuit عشان أحرقه على ال Microcontroller .

Compile Time Binding

انت ال assembly بياخد ال object file ويطلع object file .. ال object file ده بيطلع عند addresses virtual مش معروفة .. ييجي بعد كدا ال linker يحط ال instructions عند addresses بتعادل اللي موجودة في ال physical .. فلما

نيجي ن burn أو ن load الكلام ده في ال physical هيتحط عند ال Addresses بتاعت ال physical اللي أنا المفروض مظبط عليها .. وده بقا اللي احنا بنسميه compile time binding .

map.

فيه حاجه اسمها ال map file .. ايه هو دا ؟ .. ده احنا لو عايزين بنباصيه للـ linker .. ال map file ده بيبقا فيه symbols .. انت طلعت binary أو executable .. ال executable اللي انت طلعته ده بيبقا فيه سكاشن .. كل سكشن بيبقا جواه symbols .. ال symbols دي أي حاجه متعرفة بتستخدم في الكود بتاعك ده symbol .. هناخد الكلام ده في ال session الجاية ان شاء الله .. ال symbols دي ليها symbol table بيجمعها .. هو ال map file ده بيبقا فيه كل section وبيبقا فيه كل ال symbols دي .

عملية ال resolving بتتم عن طريق ال linker .. لكن اللي بيحتوي على المعلومات اللي طلعت من العملية دي بقا هو ال map file .

ال linker بقا بيجمع ال txt. بتاع ال objects كلها مع بعض في txt. واحد وال data. مع بعضيها وال bss. اللي هي uninitialized data مع بعضيها وطبعاً حط حاجات بقا لل debug والكلام ده .. وطبعاً هيحط كل حاجه عند ال Address الصح ليها في ال physical .. وهو هيحط الكلام ده بناءاً على ايه ؟ .. بناءاً على ال linker script .

كدا الحمد لله احنا خلصنا ال Lecture دي .

حابين بس نقول حاجه .. ال Embedded C مهم جداً لأن هو ده اللي هنبني عليه اللي جاي .. لو انت فاهم Embedded C قوي هتخش في أي انترفيو وانت واثق من نفسك .. Avelabs قعدت مع واحد 3 ساعات ونص في الانترفيو .. كان ساعتين وربع مش بيشتغل فيهم غير Embedded C على ال linker وال relocation وال objdump وال symbols والسكاشن .. قعد بس في الحتة دي كتير جداً .. فعشان تسد كويس لازم تبقا انت فاهم كويس .

طب احنا اللي خدناه النهاردة ده المفروض نذاكره ازاي ؟ .. أول حاجه تعملها تحل الكويز .. بس بعد ما تذاكر كويس قوي .. بعد الكويز شوف اللاب واعمله بإيدك ونزل ال tools وابدأ اشتغل على proteus .

تاني حاجه اكتب ال Platform_Types.h .. عايزك تكتب application وتستخدم ال terminal ان انت ت compile وتطلع كل حاجه .. عشان ايدك تمشي بس في الموضوع .