Proton[©] Analyser for complex and multi-file projects

Harm de Vries (hadv215 on the forum)

Introduction - how it all began

John Barrat (JohnB on the forum), creator of, a.o, the Fuse Configurator, Library Manager and P-RTOS, posted a message on the Proton[©] PDS forum, considering a tool to help him get oversight, and insight, in big, mulit-file, projects. <u>http://www.protonbasic.co.uk/showthread.php/66736-Working-with-big-projects-in-PDS</u> After exchanging some ideas with John, I started working on it.

He sent me the sources of the P-RTOS project. As test material...(30.000+ lines). Then I found out that this was full of pre-processor statements. He offered to build a dedicated program as a front-end for mine. When he went on holiday I decided to take a crack at it.

What does it do & why would you want to use it?

To start with the latter: if you have projects containing more than one file, it is sometimes hard to keep oversight of what is where. Same if you have large programs with a lot of variables and subroutines (although the Explorer in the IDE does certainly help).

What it does is the following:

- it reads through all the sources that are used in your project
- it picks up
 - all variables and symbols
 - all labels
 - all goto and gosub and their variants, including ASM instructions
 - all macros and \$defines

- it records them together with file- and line information of the declaring source Then it goes all over through the sources again to find out where all items are being referenced.

User interface.

The results are presented as follows:

- at the left a tree with all

- variables and symbols
- routines, isr's and labels
- macros and defines

all with

- the file- and line information of the declaring source

- the reference count (not for labels that are not referenced of course) Sorted in alphabetical order.

The analyser uses a separate program, the Navigator, to allow you to navigate through the Program Flow (when run in stand alone mode) or the open file in the PDS editor (when run in plugin mode).

In the first picture you see the analyser in stand alone mode.

The navigator is on the far left and can be dragged where you want it to be.

(Both the analyser and the navigator store their position and size when closing).

The top line has a different color because this is the line that references the declaration of the item you have selected.

The left panel of the analyser window holds a list of 'items' (variables, symbols, labels, routines, ISR's, macro's and \$defines).

'Items' are selected by clicking the name of the item.

The middle panel holds the Program Flow. Here all items, except the top one, are clickable.

The upper right panel holds the file structure. Clicking an item will open it in the editor of your choice (see Preferences for that).

The lower right panel shows detailed information of an 'item' or a node in the Program Flow. This is updated as you 'hover' over either one of the panels.



In the next picture you see the analyser when used as a plugin. The main difference in plugin mode is that navigation is in the PDS editor window. Selecting a line in the navigator highlights the corresponding line in the editor window.

301: EURICU-Projectivity Fit: Edit Upon Home Fit: Edit <th>Navigator - 0.1.1.7</th> <th>Proton IDE - E:\PIC\PicProjects\Tijdklok\TijdKlok.bas</th> <th></th>	Navigator - 0.1.1.7	Proton IDE - E:\PIC\PicProjects\Tijdklok\TijdKlok.bas		
11 : E VIC/DeProjectivitydes 32 : E VIC/DeProjectivitydes 33 : E VIC/DeProjectivitydes 33 : E VIC/DeProjectivitydes 33 : E VIC/DeProjectivitydes 34 : E VIC/DeProjectivitydes 35 : E VIC/DeProjectivitydes 36 : E VIC/DeProjectivitydes 36 : E VIC/DeProjectivitydes 37 : E VIC/DeProjectivitydes 38 : E VIC/DeProjectivitydes 39 : E VIC/DeProjectivityd	307 : E:\PIC\PicProjects\Tijdklo	File Edit View Help		
334 : EXP(C/R/PojecsX1)jude Comple and Rogan Population in the set of the s	215 : E:\PIC\PicProjects\Tijdklo	New 🗞 Onen 🗆 Save 🚶 Ort 🗅 Core 🗅 Parte 😥 Undo 😋 Redo 🚴 Print 😤 Pint 🍄 Renjace 🔯 Indent 🐺 Ortident 🗮 Comment 🗮 Uncomment		
D3: E:UP(C)CPOPERSUNDED Complete Complete Ad Program Program Ref Image: Program Image: Program Ref Image: Program Image: Program Image: Program Image: Program Image: Program	334 : E:\PIC\PicProjects\Tijdklo			
Image: Second	335 :: E:VIC://projects/Tijdkier 👋 Compile and Program 👋 program 👋 analyser			
Image: Second		I TjdKlok		
Image: Second	File P	Code Explorer - 🔀 TijdKlok	🕰 • 🔀	
Image: State of the state		B 18F1320 ^ 259 EndIf	A	
Image: Construction of the second		260 EndIf		
Image: Control Series of the state of t		Declares 261 EndIf		
Image: State of the state	÷-	Create_Coff E 262 ; switch off		
Image: Second		D Xtal 263 If iHou = iHouSwOff Then		
Image: Based 265 Endif 266 Endif 267 Endif 268 If iMin = 10 Then 269 If iMin = 10 Then 260 If iMin = 10 Then 261 If iMin = 10 Then 263 If iMin = 10 Then 264 If iMin = 10 Then 265 If iMin = 10 Then 266 Doskub GetOPS 271 Oosbub GetOPS 272 BenIff 273 BenIff 274 RodIff 275 Goto MainLoop 275 Goto MainLoop 275 Retat 276 Ratic 276 Retat 277 Getabat <th></th> <td>D FSR_Context_Save</td> <td></td>		D FSR_Context_Save		
Herein USTA HEREIN HEREI		Berial Baud		
Image: Marked State 268 Image: Marked State 268 Image: Marked State 268 Image: Marked State 268 Image: Marked State 270		BindIf Histial_RCSTA		
Image: Second		- D Hserial TXSTA 268		
Image: State of the second		B Hserial_Clear 269 If iMin = 10 Then		
Image: Second		270 If iSec = 0 Then		
GetGPS		Adin_lad 271 GoSub GetGPS		
GetCPS Valiable Vali		272 bSkipInc - 1		
W atter 275 GoTo MainLoop 276 GoTo MainLoop 276 277 W Merk 276 278 LDR + ADIn 0 279 Return 280 281 281 282 282 IncrementClock: 281 282 282 Inc 1Sec W Nk 282 281 282 282 Inc 1Sec W Nk 282 281 28 282 Inc 1Sec 280 281 281 282 282 Inc 1Sec 283 100 284 100 284 100 282 100 282 100 284 100 284 100 284 100 285 100 286 100 287 100 288 100 289 100 280 100 281 100 282 100 283 100 284 100 284 100 284 100 <	GetGPS	273 Endir		
Image: Sources: 2276 Containing Image: Sources: 2278 Containing Image: Sources: 228 Image: Sources: Image: Sources: Image: Sources: Image: Sources:		Wawerk 275 GoTo MainLoon		
Image: Constraint of Defermine Confermine Conferm		- W iWerk 276		
Image: Content of Participation of Parti		aNext 277 GetAD:		
Image: State of the second		- V aDate 278 LDR = ADIN 0		
Waime 280 281 IncrementClock: 282 IncrementClock:				
Inc 281 Increment/Clock: 282 Inc 1Sec * 284 Inc 2Sec * 294 Inc 2Sec * 295 Inc 2Sec * 296 Inc 2Sec *				
Inc 1580 282 1nc 1580 * Read/S * * * Read/S * * WHT: Reserving 12 bytes at the top of RAM for High Interrupt Context * Success: 2200 program bytes used, 222 variable bytes used In 271 : Col 1 GetAD: * Increment/Clock: *		- V dx 281 IncrementClock:		
Revite Rev		282 Inc 1Sec	*	
Renults		The	•	
HMT : Reserving 12 bytes at the top of RAM for High Interrupt Context Success : 2280 program bytes used, 222 variable bytes used Interrupt Context GetAD: Get		Results	- 🛛	
Success: 2280 program bytes used, 222 variable bytes used In 271 : Col 1 GetAD: GetAD: ConcernmentClock- ConcernmentClo	WINT: Reserving 12 bytes at the top of RAM for High Interrupt Context			
Success: 2280 program bytes used, 222 variable bytes used In 271 : Col 1 GetAD: ColorementClock:				
GetAD: Increment/lock:				
GRAD: Increment lock: + Increment lock: + + + + + + + + + + + + + + + + + + +	Vicess : 2200 program bytes useq, 222 vanable bytes useq 🔤 Ln 2/1 : col1		a,	
		GetAD:	-	
		E-IncrementClock-	b	
Pope cussing in plugin mode				
	pone, raining in plugin mode	jozoo mseus, orz lines		

Now it may be that you select a line in an include file. If it is opened in a tab in the IDE, this tab will become the 'current' tab. If it is not opened in the IDE, it will be loaded and it will become the 'current' tab. After closing the analyser you will be asked if you want the files opened by the analyser to be closed.

About the Program Flow. This is a tree, starting at the beginning of the first file. It shows all labels, Goto's and Gosub's. Note that e.g. Branch is treated as a Goto.

What it also shows is the 'indirect' GoSub structure. Say, you have a routine ABC that calls DEF and this one calls GHI. When Gosub ABC is encountered, DEF and GHI are inserted in the flow, looking like this aLabel1: Gosub ABC

Gosub ABC Gosub DEF Gosub GHI when DEF is encountered aLabel2: Gosub DEF Gosub GHI

Conditional compilation

Some of you may use it, others not.

The analyser will invoke a pre-processor when it encounters directives for conditional compilation, so only the lines that meet the conditions are passed to the analyser.

Output

All information is stored in a report file.

It can be viewed by clicking View>the filename of the report.

When the pre-processor was invoked there are two more output options:

- the generated .bas file. This is a file that contains all lines from the original source and the includes after evaluating the conditions. You can load it in the IDE when running in plugin mode. It is saved to disk so you could also use it later.

- the tracking output of the pre-processor.

This contains, pretty technical, data on the evaluations the pre-processor performed. It is meant for those people that want to debug their pre-processor statements. (It was a great help for me when building the pre-processor, why not tidy it up and include it.)

How to use the analyser

There are two ways to use it:

- as a plugin
- as a standalone program

As a plugin:

As soon as you launch the analyser, it will ask you if this is the 'main' file of your project. Selecting the right file is important because it determines the information you get. If you have an included file in the 'current' tab in the IDE, the analyser will only process that one.

Press 'Cancel' to select your 'main' file as the current tab.

The popup can be disabled through the Preferences menu.

As a standalone program:

Start the analyser, click File>Open to select the file you want to process. Again you are asked to confirm if this is a 'main' project file.

Preferences

There are three preferences.

The first one is easy: you can select the program that will be used if you click a filename.

The second is a bit more complex and needs some explanation.

Consider the phenomena 'Define' and 'Macro'.

Both are is essence nothing more than text-replacing techniques.

In P-RTOS, JohnB has a \$Define 'Forever' that reads 'Until 1 = 0'.

A Macro is like building your own function, using parameters.

Every time the compiler encounters a reference to a \$Define or Macro, it just picks up the text and inserts it in the source.

You would not like to see 'Forever' in the window with the program flow (because it does not alter the flow).

But a \$Define or Macro might contain one or more Gosub's.

And that is part of the program flow. Sometime you might want to see that, sometimes not.

The preference 'Show invoking Define/Macro' has three options:

- Always

- Never
- Only when it contains GoSub/Call

If you change the 'Show Define and Macro' the program will repeat the analysis of the program flow to reflect the new value. Enabling/disabling options can also be done using a popup menu (right click on the program flow).

This popup menu also gives you the opportunity to hide/show unreferenced labels in this tree.

The third relates to the warning popup regarding the 'main' file. You can disable or enable it here.

Notes

1: Make sure your program compiles. The $\mathsf{Proton}^{\mathbb{C}}$ Analyser may crash if your program contains errors.

2: The syntax of the Basic language does not prevent recursive calling, but a microcontroller will most certainly reset itself.

To prevent the analyser from crashing, a maximum of 16 stack levels is allowed, after that it just stops and shows a warning.

Known errors

At this moment there is one known error. And I don't have a clue of how to fix it. When a Gosub is encountered, the Proton[©] Analyser tries to locate a label 'above' the Gosub in order to have in the correct place in the program flow window. Consider this:

label1:

```
statements
if <condition> then Return
more statements
Return
```

Gosub A_Routine

```
Main_program_loop:
Gosub label1
Goto Main_program_loop
```

Now clearly, "Gosub A_Routine" will never be executed, but there's no way the analyser will 'know' that. Checking a 'Return' in a previous line will not help, because it may be a conditional return as in the example above. Please understand, the analyser works from top to bottom, not backwards.

At this moment, the Proton[©] Analyser just considers it to be part of the routine called "label1".

Technical details

Files:

- Main program: ProtonAnalyser.exe

- Navigator: ProtonAnalyserNavigator.exe. This won't run as a separate program, it must be called by the analyser.

Preferences:

- ProtonAnalyserPreferences.txt (please don't do manual changes, they might not work).

Output directory:

_Output: This directory is created in the directory of your main program. It contains all output.

Final words

Thanks to Roger and Peter for sending me input. It was disturbing how easy it is to forget the differences in programming style and the effects it has on this program. And of course thanks for John for the interesting email-discussions we had over this project, his input, the icons, his offer to build a dedicated pre-processor (that he didn't have to build in the end), the help with the plugin api and the help file. But most of all for the code he sent me for a critical part of this project. Last, but not least, thanks Lester for PDS.